МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств ВТ и автоматизированных систем»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему

Синтаксический редактор телеметрической информации

Студент 03.06.20 Прокуров Дмитрий Алексеевич
(подпись, дата)

Руководитель 03.06.2020 Федюнин Р.Н.
(подпись, дата)

Нормоконтролёр _____ А.В. Кучин
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 03.06.2020 г. № 12)

Заведующий кафедрой _____ М.А. Митрохин
(подпись)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 17.06.20 № 8)

Секретарь ГЭК _____ Е.И. Гурин
(подпись)

Пенза 2020

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«30» апреля 2020 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

1. Студент Прокуров Дмитрий Алексеевич гр. 16вв2 факультета вычислительной техники.

2.Тема работы «Синтаксический редактор телеметрической информации».

Тема утверждена приказом ПГУ №_ 0335а/о___ от "_30__"_____апреля_____2020___г.

3. Руководитель выпускной квалификационной работы бакалавра к.т.н., доцент каф ВТ ПГУ Федюнин Р.Н.

4. Задание на работу (назначение разработки, исходные данные и т.п.).
Назначение разработки- приложение, предназначенное для редактирования файла представления телеметрии.

5. Перечень подлежащих разработке вопросов.
Требуется произвести анализ предметной области, определить архитектуру приложения, выбрать язык программирования для приложения, выбрать способ взаимодействия между модулями.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10	23.03.20	
2.	Выбор архитектуры приложения, языков программирования, способа взаимодействия	10	04.04.20	
3.	Разработка интерфейса	20	10.04.20	
4.	Разработка алгоритмов	30	18.04.20	
5.	Тестирование и отладка	10	4.05.20	
6	Оформления выпускной работы	20	15.05.20	

Дата выдачи задания "30" апреля 2020 г.

Руководитель бакалаврской работы _____ Федюнин Р.Н.
(подпись, дата)

Задание к исполнению принял студент _____ Прокуров Д.А.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Выпускная квалификационная работа бакалавра содержит 176 листов, 61 рисунок, 5 приложений.

СИНТАКСИЧЕСКИЙ РЕДАКТОР ТЕЛЕМЕТРИЧЕСКОЙ ИНФОРМАЦИИ.

Цель работы – разработать приложение для проверки и редактирования файла представления телеметрической информации.

Объектом разработки будет являться приложение, представляющее собой редактор XML-документа, описывающего телеметрию прибора.

В результате проведенной работы был реализован проект, который удовлетворяет условию поставленной задачи, а именно, создано приложение для проверки и редактирования файла представления телеметрической информации.

Созданное приложение может быть внедрено на предприятие, где требуется обрабатывать файлы описания телеметрии.

Содержание

РЕФЕРАТ 4
Содержание 5
Введение 7
1. Анализ предметной области для решения поставленной задачи 8
1.1 Постановка задачи 8
1.2 Актуальность использования формата XML 8
1.3 Обзор бесплатных редакторов XML-документов 8
1.4 Определение архитектуры приложения 12
1.5 Выбор языка программирования и среды разработки 14
2. Разработка приложения 16
3. Описание работы приложения 19
3.1 Руководство пользователя 19
3.1.1 Назначение XML-Editor 19
3.1.2 Условия выполнения XML-Editor 20
3.1.3 Состав XML-Editor 21
3.1.4 Подготовка к работе 22
3.1.5 Описание работы с программой XML-Editor. 23
3.2 Описание разработки 51
Безопасность жизнедеятельности 64
Заключение 66
Список используемых источников 67
Приложение А Листинги программы 68
Приложение А. 1 – файл «global_vars.h» 68
Приложение А. 2 – файл «tabletree.h» 72
Приложение А. 3 – файл «xml_check.h» 74
Приложение А. 4 – файл «xml_editor.h» 77
Приложение А. 5 – файл «global_vars.cpp» 80
Приложение А. 6 – файл «main.cpp» 83
Приложение А. 7 – файл «tabletree.cpp» 83
Приложение А. 8 – файл «xml_check.cpp» 106
Приложение А. 9 – файл «xml_editor.cpp» 140

Приложение B Пример XML-документа 165
Приложение C UML-диаграммы приложения 166
Приложение C.1 – UML-диаграмма классов 166
Приложение C.2 – UML-диаграмма вариантов использования 166
Приложение C.3 – UML-диаграмма последовательности 167
Приложение D Результаты работы программы 168
Приложение D.1 – Результат загрузки файла 168
Приложение D.2 – Добавление элемента 169
Приложение D.3 – Редактирование элемента 170
Приложение D.4 – Сохранение документа 171
Приложение E Презентация 172

Введение

В природе существует огромное количество постоянно изменяющихся величин, для описания которых необходимо следить за изменением их параметров. С развитием средств вычислительной техники совершенствовались способы получения, обработки и представления телеметрической информации. В настоящее время телеметрия применяется во множестве областей человеческой деятельности, таких как сельское хозяйство, медицина, вендинг, оборонная и космическая отрасли.

Одним из новейших средств представления телеметрической информации является язык XML. XML – расширенный язык разметки, разработанный для создания, обработки и хранения информации в понятном для программ и человека в виде.

В данной работе поставлена задача разработать синтаксический анализатор телеметрической информации, представляемой в форме XML-документа и формируемой с помощью другой программы – чёрного ящика.

1. Анализ предметной области для решения поставленной задачи

1.1 Постановка задачи

Необходимо разработать приложение для редактирования XML-документа, описывающего телеметрию. Приложение должно обладать простым и интуитивно понятным пользовательским интерфейсом и реализовывать функционал загрузки, проверки и редактирования XML-документа, представляющего описание телеметрии.

- Платформа: ОС Windows.
- Среда разработки: QT Creator.
- Язык разработки: C++.

1.2 Актуальность использования формата XML

Язык XML – расширенный язык разметки – является универсальным языком описания документов и позволяет представлять сложные иерархические структуры данных в виде структуры тэгов. Язык XML одобрен Консорциумом Всемирной паутины и удобен как для программной обработки, так и для чтения человеком. В настоящее время широко распространён в качестве основы для форматов представления данных, например, для формата FB2. Технология, используемая для программной обработки документов XML, называется DOM - Document Object Model, представляющая тэги XML-документа в виде объектов с определённым набором свойств.

1.3 Обзор бесплатных редакторов XML-документов

Рассмотрим возможные аналоги редакторов XML-документов, для того чтобы лучше понять, что необходимо для реализации поставленной задачи.

В первую очередь нужно обратить внимание, что для заданной конкретной предметной области, а именно, редактирования документа,

представленного в определённом формате, редакторы отсутствуют. Поэтому рассмотрим иные программные средства, позволяющие редактировать произвольные XML-документы.

- Exchanger XML Lite

Exchanger XML Lite является мощным программным обеспечением со много функциональностью, которая является в основном редактором. При помощи этого программного обеспечения пользователь может отредактировать, просмотреть, управлять и преобразовать документы XML.

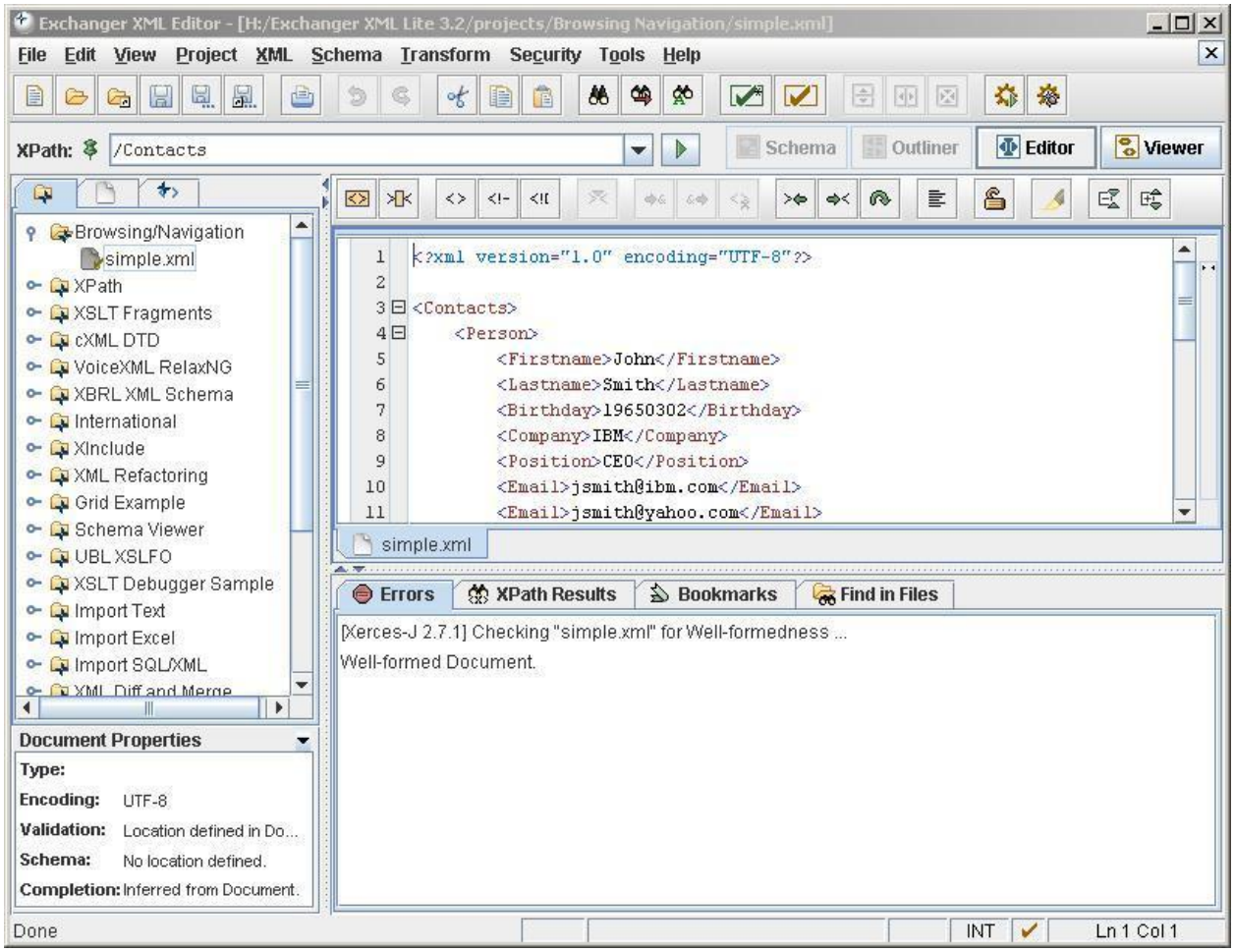


Рисунок 1 - скриншот Exchanger XML Lite

- EditiX

Редактор, обладающий широким функционалом. Умеет делать XSLT трансформации. Подсказки выдаются в виде выделения похожих частей кода, а не выпадающим списком и табличного представления документа.

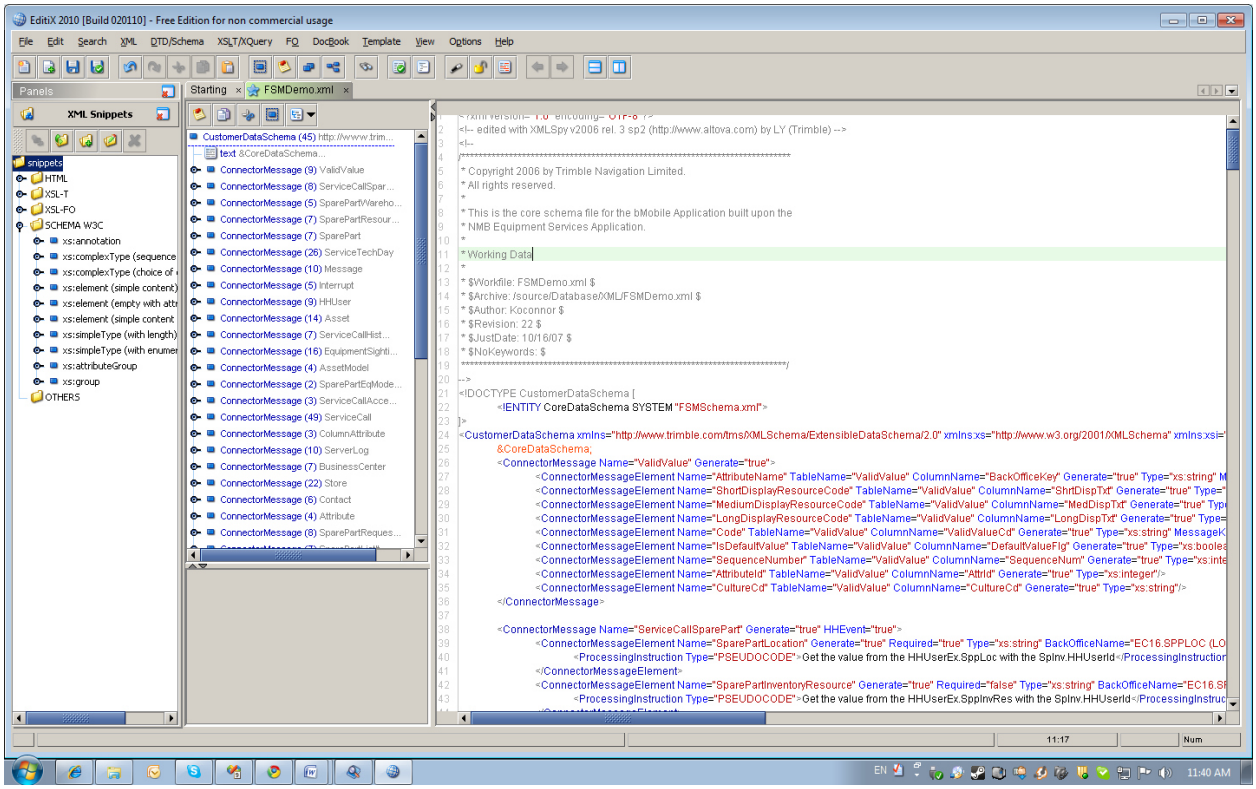


Рисунок 2 - скриншот EditiX

- XML-Maker

Показывает XML в табличном виде доступном для редактирования. К сожалению, нет подсказок и возможности делать XSLT трансформации.

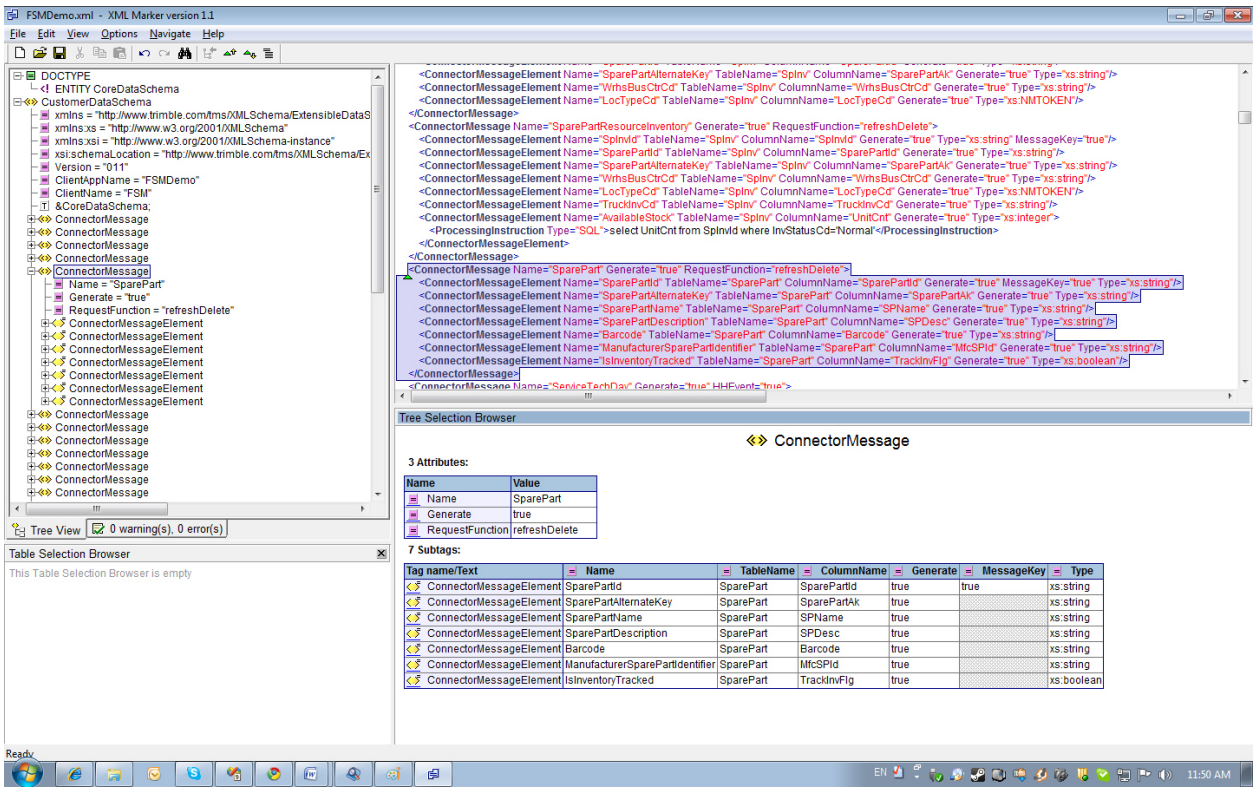


Рисунок 3 - скриншот XML Marker

- XMLPad

Показывает XML в табличном виде и в виде дерева. Есть подсказки. Есть возможность делать XSLT трансформации.

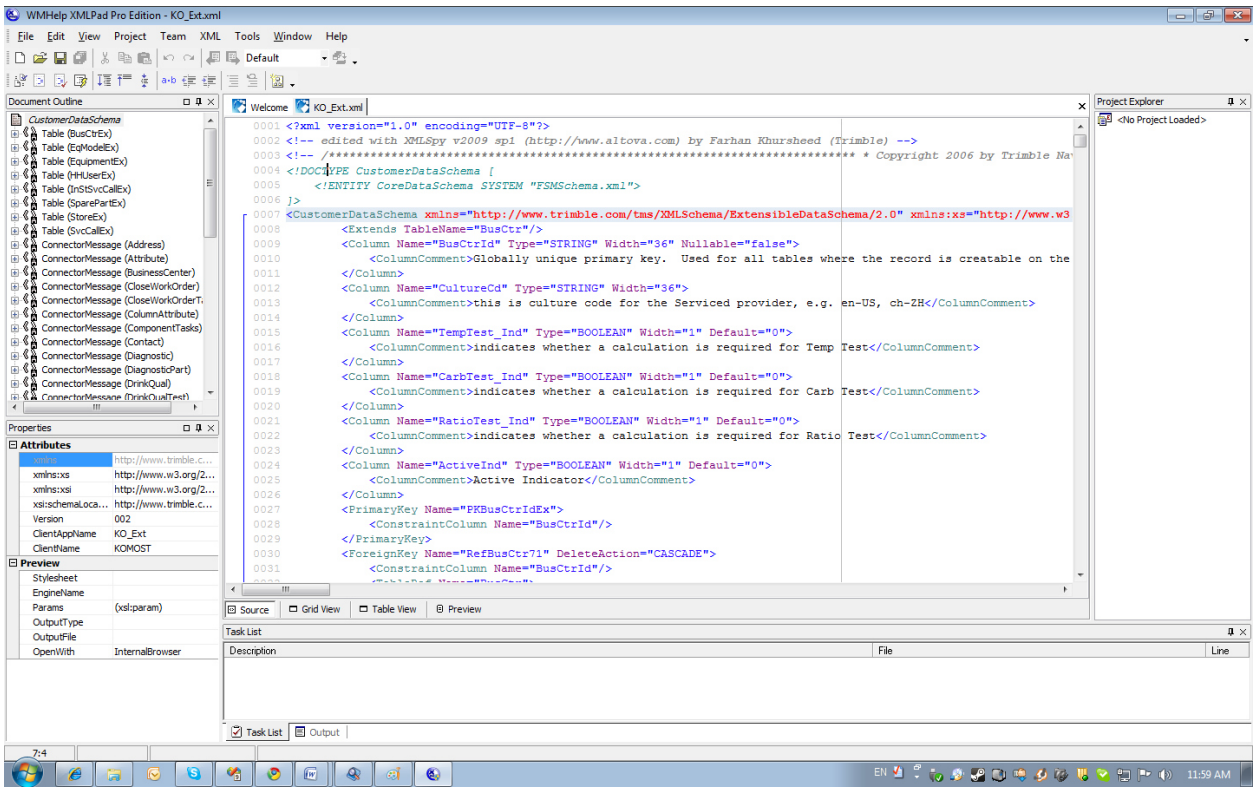


Рисунок 4 - скриншот XMLPad

1.4 Определение архитектуры приложения

Исходя из требования к задаче, была выбрана архитектура MVC. Это одна из самых распространенных архитектур. Архитектура MVC позволяет нам разделить код приложения на 3 части: Модель (Model), Вид или Представление (View) и Контроллер (Controller). Впервые она была описана в 1978 году, и предназначалась для приложений с графическим интерфейсом (окошками и кнопками), но позже была адаптирована и для веб-приложений.

Разделение на части позволяет упростить большой по объему код. Если код писать одним длинным скриптом, в нем становится тяжело разобраться, и тяжело вносить изменения, не допустив ошибку.

MVC не привязана к какому-то конкретному языку программирования, и не требует использования объектно-ориентированного программирования или какой-то другой парадигмы.

Разделение на части здесь не значит, что в коде должно быть ровно 3 файла (или 3 папки с файлами, или 3 класса) с названиями model, view и controller. MVC ничего не говорит нам по поводу того, как организовывать файлы с кодом. На практике модель часто занимает основной объем приложения, и представлена в виде большого числа разнотипных классов - сущностей, сервисов, классов работы с БД, и для каждого вида классов делают отдельные папки.

MVC применима к разным видам приложений - и к серверным веб-приложениям, и к десктопным (клиентским) приложениям.

Модель содержит в себе всю логику приложения, она хранит и обрабатывает данные, при этом не взаимодействуя с пользователем напрямую (обратиться к Модели можно только из кода, вызывая ее функции). Представление отображает данные, которые ему передали. Контроллер отвечает за выполнение запросов, пришедших от пользователя.

Плюсы:

- Единая концепция системы. Несомненным плюсом MVC является единая глобальная архитектура приложения. Даже в сложных системах, разработчики (как те, которые разрабатывали систему, так и вновь присоединившиеся) могут легко ориентироваться в программных блоках. Например, если возникла ошибка в логике обработки данных, разработчик сразу отбрасывает 2-блока программы (controller и view) и занимается исследованием 3-го (model).
- Упрощен механизм отладки приложения. Т. к. весь механизм визуализации теперь сконцентрирован в одном программном блоке, упростились механизмы опционального вывода графических элементов.

Недостатки:

- Усложнен процесс расширения функционала. Недостаточно просто написать функциональный модуль и подключить его в одном месте программы. Каждый функциональный модуль должен состоять из трех частей, и каждая из этих частей должна быть подключена в соответствующем блоке.
- Необходимость использования большего количества ресурсов. Сложность обусловлена тем, что все три фундаментальных блока являются абсолютно независимыми и взаимодействуют между собой исключительно путем передачи данных. Controller должен всегда загрузить (и при необходимости создать) все возможные комбинации переменных и передать их в Model. Model, в свою очередь, должен загрузить все данные для визуализации и передать их во View.

В нашем случае приложение будет разделено на 3 модуля: модуль интерфейса, модуль данных, модуль XML.

1.5 Выбор языка программирования и среды разработки

Исходя из требований к задаче, необходимо выбрать язык программирования для реализации редактора. После чего на основе выбранного языка нужно определить среду разработки.

Приложение должно обладать графическим интерфейсом и уметь реагировать на действия со стороны пользователя. Т.к. это является одним из ключевых требований, то необходимо выбрать язык программирования, который позволяет относительно легко создавать графический интерфейс и поддерживать механизм событийности. Также необходимо, чтобы была возможность использовать классы при написании программы. Это повысит читаемость исходного кода и позволит выделить сущности необходимые для грамотной реализации программы. В свою очередь, это облегчит поддержку и развитие приложения.

С учетом выше сказанного был выбран язык программирования C++. C++ является объектно-ориентированным языком программирования, в котором присутствует такое понятие как классы. Классы позволяют сокрыть данные и определить набор операций, выполняемых над ними. За счет этого можно относительно легко разбить программу на сущности, которые будут представлять собой независимые части. Это повысит наглядность и легкость восприятия кода программы, что в свою очередь поможет проще обнаружить и локализовать возможные ошибки в исходном коде программы.

Графический интерфейс приложения можно реализовать с помощью средств платформы QT, таких как QT Widgets. Данный инструмент позволяет относительно легко создавать графические интерфейсы за счет того, что имеется графический редактор формы, который позволяет просмотреть её вид на этапе разработки непосредственно до запуска программы. Данный редактор автоматически генерирует весь код, отвечающий за внешний вид, для каждого элемента формы, что позволяет ускорить разработку интерфейса и избежать нежелательных ошибок в коде, отвечающем за интерфейс. Помимо этого, редактор QT Widgets позволяет использовать в создании графического интерфейса пользовательские компоненты интерфейса, что позволит значительно упростить разработку интерфейса.

Выбор среды разработки напрямую зависит от выбранных языков программирования. Т.к. для реализации приложения был выбран язык C++, то в качестве среды разработки была выбрана среда QT Designer. QT Designer – это бесплатная интегрированная среда разработки с мощными, эффективными возможностями для кодирования, инструментами кроссплатформенных разработок мобильных приложений для Windows, iOS, Android, веб- и облачных приложений, а также широкие возможности для работы с такими расширениями, как OpenGL и QML.

2. Разработка приложения

После запуска программы отображается интерфейс приложения. В момент запуска в приложение ещё не загружен документ, поэтому пользователь может либо создать документ, либо загрузить файл, содержащий описание телеметрии на языке XML. После загрузки документа, он проверяется на соответствие формату документа, после чего программа предупреждает пользователя о наличии или отсутствии ошибок в документе. Если ошибка есть, то программа предупреждает пользователя, в каком месте документа она совершена, и какого эта ошибка рода. В случае, если документ не содержит ошибок, его структура отображается в левой части приложения в виде дерева. Пользователь имеет следующие возможности работы с узлами дерева: создание дочерних узлов, удаление, перемещение и добавление «братских» узлов. В случае, если пользователь осуществляет действие, которое приведёт к нарушению требований к формату телеметрии, программа выведет сообщение, предупреждающее пользователя о невозможности совершения подобных действий.

В правой части экрана расположен редактор свойств узла. Свойства узла имеют строго определённый спецификацией формат представления данных, и не могут принимать значения, которые противоречат формату представления данных. При выборе пользователем узла дерева в левой части приложения, в правой части будут отображаться его свойства. Для того, чтобы при редактировании документа свойства не могли принять неправильные значения, ввод данных осуществляется с помощью визуальных компонентов интерфейса, ограничивающих возможности пользователя по изменению этих свойств.

UML-диаграмма последовательности представлена на рисунке 5.

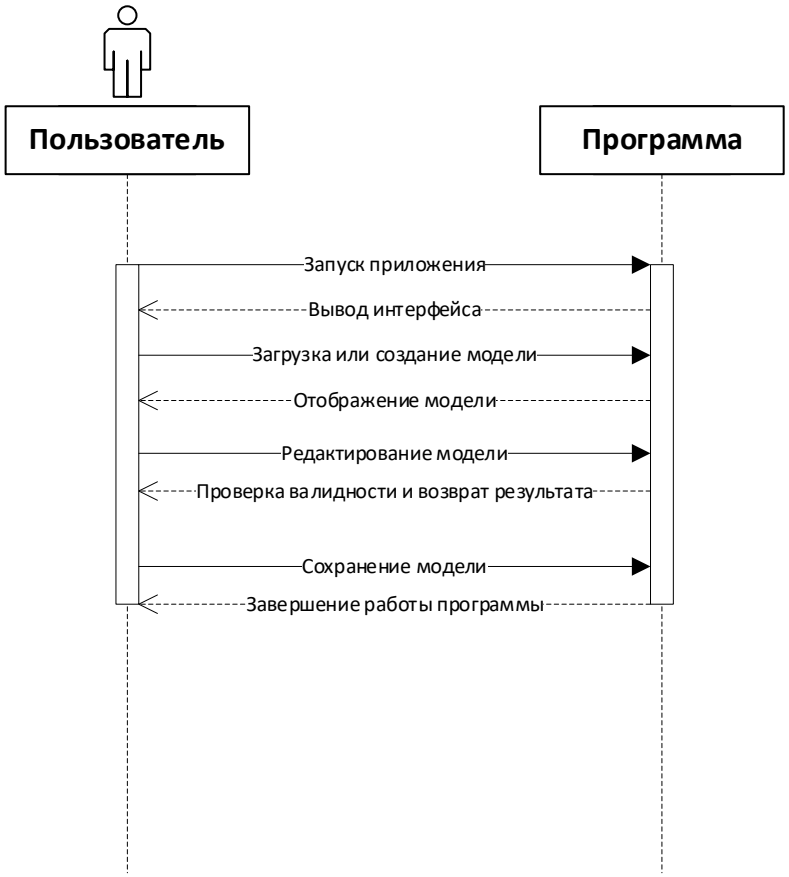


Рисунок 5 – UML-диаграмма последовательности

Рассмотрим модули, используемые в программе:

global_vars – модуль, представляющий собой набор глобальных констант и переменных, таких, как названия узлов, граничные значения, а также набор сообщений.

TableTree – класс интерфейса таблицы представления и редактирования свойств узла документа.

XML_Check – статический класс, предоставляющий набор методов для работы с XML-документом, включая его проверку и редактирование

XML_Editor – класс интерфейса, в котором реализована обработка действий пользователя и взаимодействие с UI.

На рисунке 6 изображена UML-диаграмма классов программы. Она показывает связи между классами и модулями программы.

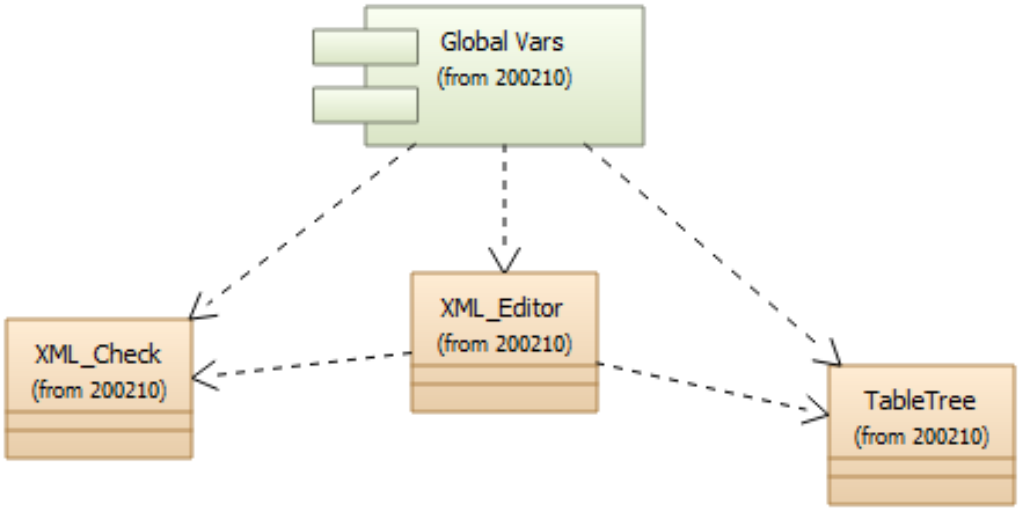


Рисунок 6 - UML-диаграмма классов

UML-диаграмма вариантов использования представлена на рисунке 7.

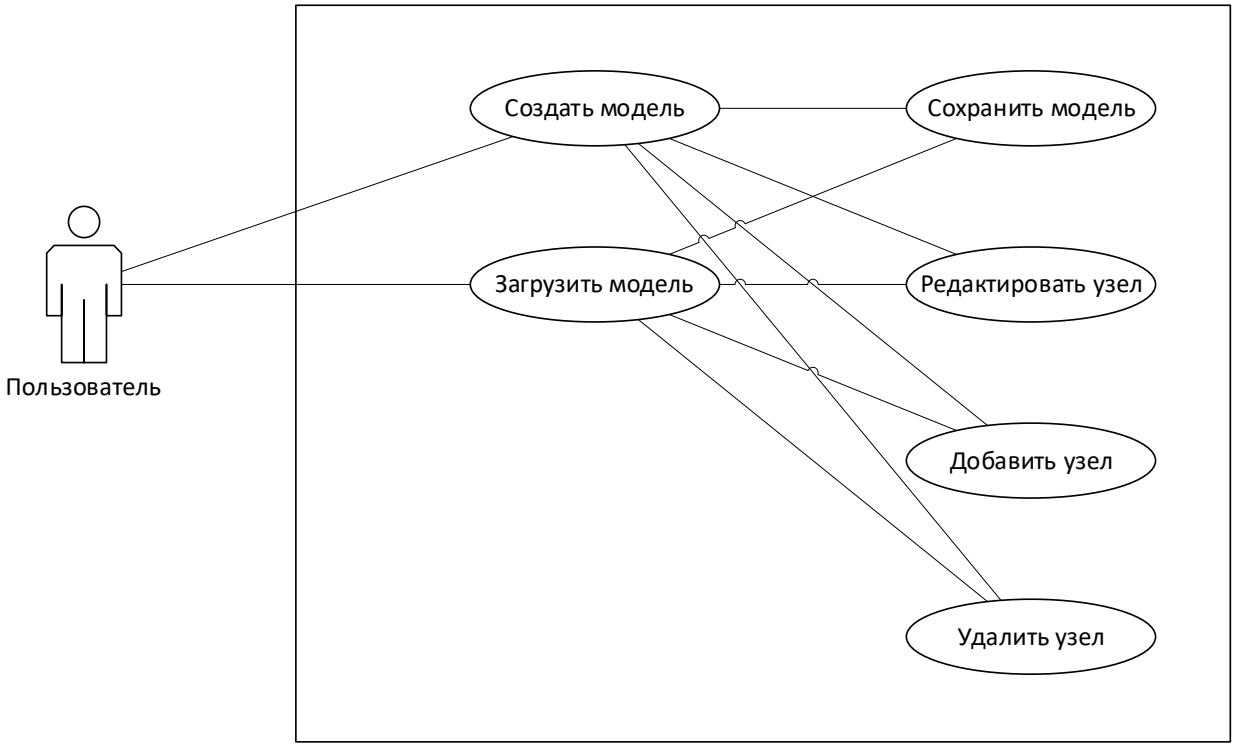


Рисунок 7 - UML-диаграмма вариантов использования

3. Описание работы приложения

3.1 Руководство пользователя

3.1.1 Назначение XML-Editor

3.1.1.1 Приложение XML-Editor предназначено для представления XML-документа, описывающего телеметрию, в виде дерева и обеспечивает возможности изменения, добавления и удаления полей телеметрии.

3.1.2 Условия выполнения XML-Editor

3.1.2.1 ПО XML-Editor может быть использовано самостоятельно. В ПО предусмотрены возможности для создания XML-документа, описывающего телеметрию.

3.1.2.2 Для выполнения XML-Editor необходима операционная среда Windows XP, 7, 8 или 10.

3.1.2.3 Значения параметров, указанные на рисунках, приведены в качестве примера. В приложении В содержится пример используемой в данном руководстве модели телеметрии.

3.1.2.4 Для работы программы необходимы следующие минимальные характеристики ПК:

- Частота процессора: 1 ГГц
- ОЗУ: 512 Мб
- 4.5 ГБ свободного места на жёстком диске

3.1.3 Состав XML-Editor

3.1.3.1 В состав XML-Editor входят следующие элементы:

- ПО XML-Editor;
- Данное руководство пользователя.

3.1.4 Подготовка к работе

3.1.4.1 Для подготовки ПО к работе необходимо загрузить ОС Windows и запустить исполняемый файл приложения “XML-Editor.exe”.

3.1.5 Описание работы с программой XML-Editor.

3.1.5.1 Описание пользовательского интерфейса

3.1.5.1.1 Общие сведения

Работа оператора с ПО XML-Editor ведется с помощью мыши и клавиатуры.

Разрешение экрана не влияет на работу программы. Допускается изменение пользователем размера главного окна программы, работа в полноэкранном режиме. На рисунке 8 показано основное окно программы после запуска. Минимальный размер основного окна: 622:236 пикселей.

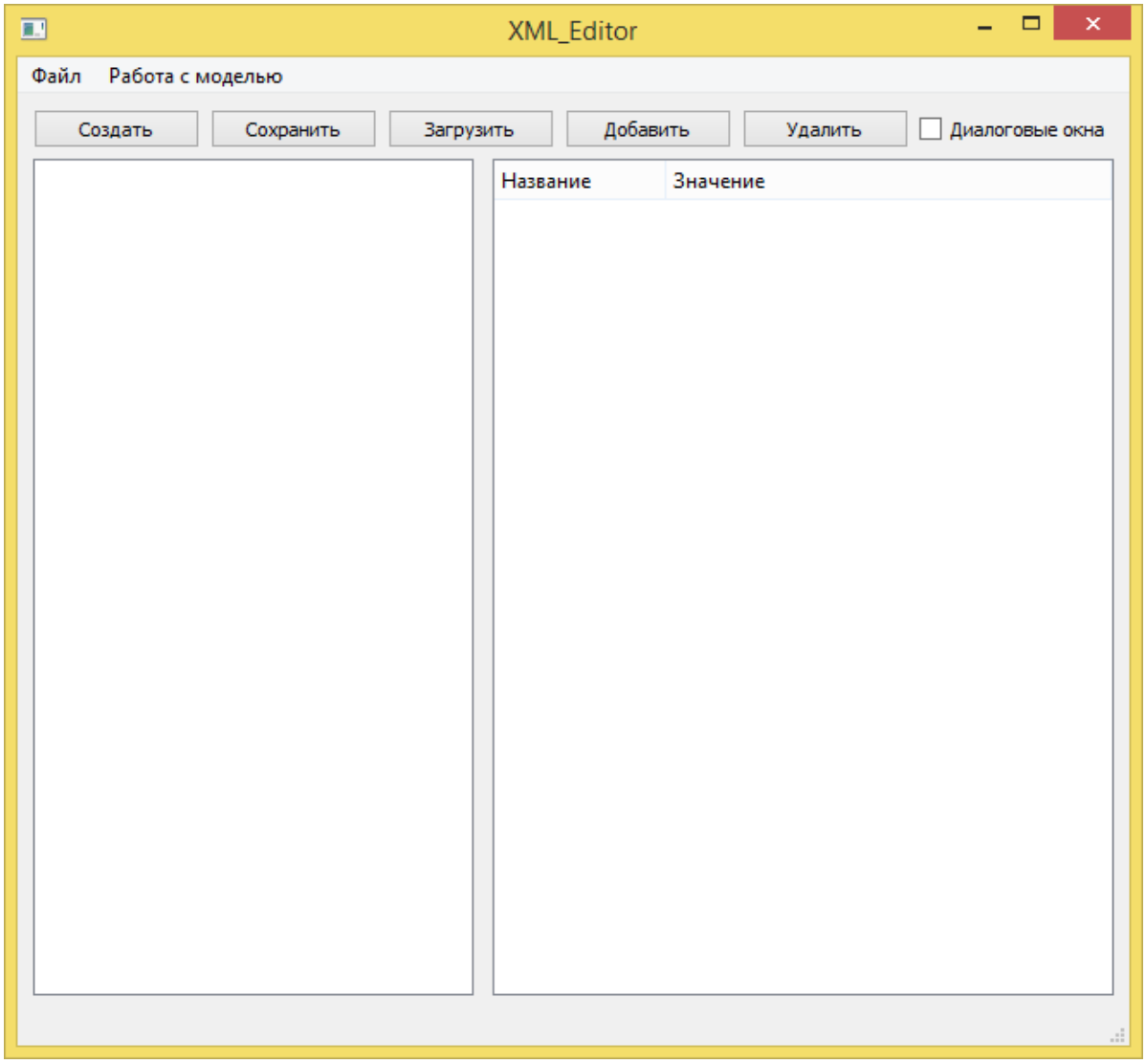


Рисунок 8 - Стандартный размер окна программы

На экране окна программы расположены:

- Панель меню
- Панель кнопок быстрого доступа
- Флаг отображения вспомогательных диалоговых окон
- Панель дерева
- Панель таблицы

3.1.5.1.2 Панель меню

Панель меню расположена в верхней части главного окна приложения (рисунок 9).

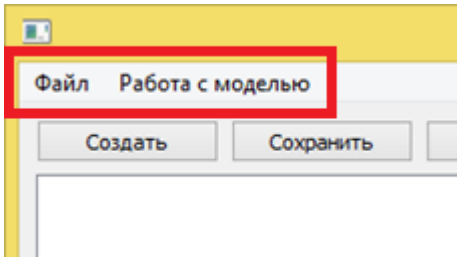


Рисунок 9 - Панель меню

Панель состоит из следующих элементов:

- Пункт меню «Файл» (рисунок 10).
- Пункт меню «Работа с моделью» (рисунок 11).

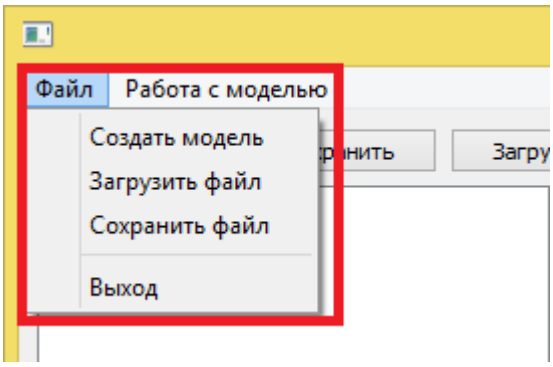


Рисунок 10 - Функции меню для работы с файлом

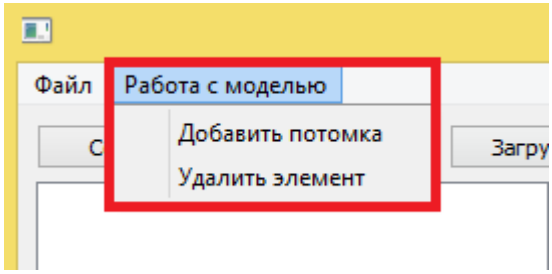


Рисунок 11 - Функции меню для работы с моделью

Подпункты меню «Файл» выполняют соответственно следующие функции:

- Создание модели (пункт 3.1.5.2.2)
- Загрузка модели (пункт 3.1.5.2.3)
- Сохранение модели (пункт 3.1.5.2.4)

Подпункты меню «Работа с моделью» выполняют соответственно следующие операции:

- Добавление потомка (пункт 3.1.5.2.5)
- Удаление элемента (пункт 3.1.5.2.6)

3.1.5.1.3 Панель кнопок быстрого доступа

Панель кнопок быстрого доступа расположена в верхней части главного окна приложения ниже панели меню (рисунок 12).

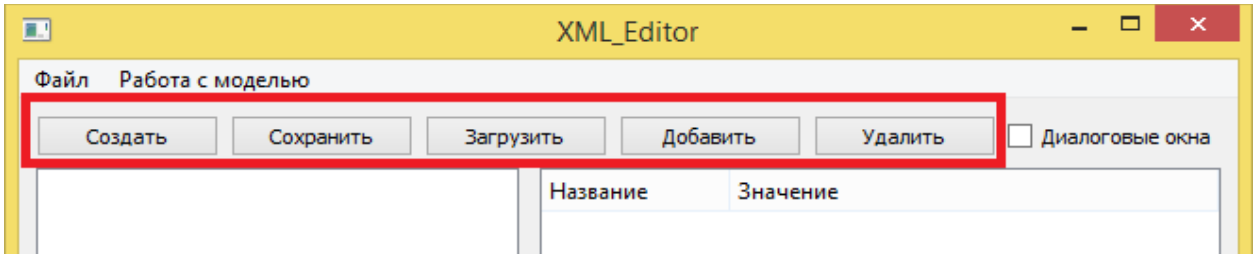


Рисунок 12 - Панель кнопок быстрого доступа

Панель состоит из следующих элементов:

- Кнопка «Создать» используется для создания модели (пункт 3.1.5.2.2).
- Кнопка «Загрузить» используется для загрузки модели телеметрии (пункт 3.1.5.2.3).
- Кнопка «Сохранить» используется для сохранения XML-документа (пункт 3.1.5.2.4).
- Кнопка «Добавить» используется для добавления выделенному элементу потомка (пункт 3.1.5.2.5).
- Кнопка «Удалить» используется для удаления выделенного элемента (пункт 3.1.5.2.6).

3.1.5.1.4 Флаг отображения вспомогательных диалоговых окон

Флаг отображения вспомогательных диалоговых окон расположен справа от панели кнопок быстрого доступа (рисунок 13) и в случае его установки при изменении каких-либо свойств (пункт 3.1.5.2.9), затрагивающих другие элементы, на экран будут выводиться сообщения, требующие от пользователя подтвердить свои намерения. Также влияет на вызов сообщения при создании нового документа (пункт 3.1.5.2.2). Примеры сообщений приведены в пункте 3.1.5.2.11.

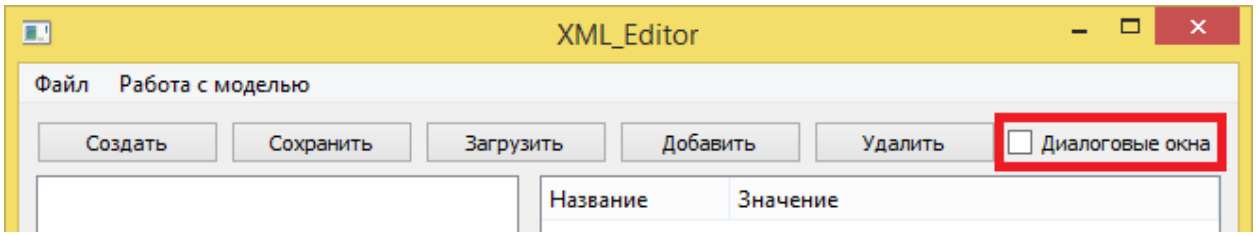


Рисунок 13 - Флаг отображения вспомогательных диалоговых окон

3.1.5.1.5 Панель дерева телеметрии

Панель дерева расположена ниже панели кнопок быстрого доступа (пункт 3.1.5.1.3) по левую сторону экрана (рисунок 14).

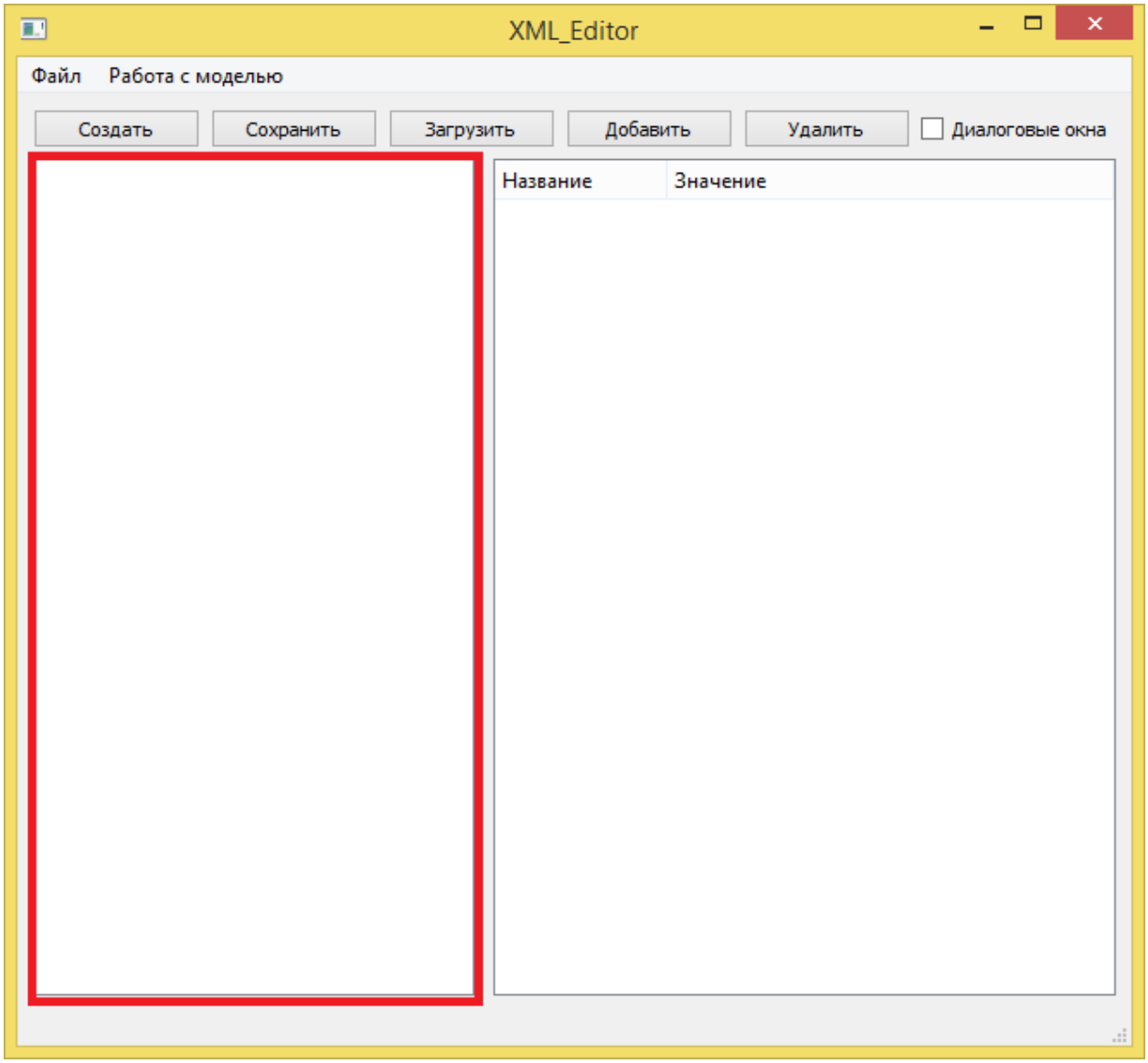


Рисунок 14 - Панель дерева

В панели отображается иерархическое представление элементов телеметрии. При выборе элемента он подсвечивается, и в панели таблицы (пункт 3.1.5.1.6) справа от панели дерева отображается набор свойств (пункт 3.1.5.2.9) элемента. Помимо этого, в программе имеется контекстное меню (пункт 3.1.5.1.7), вызываемое при клике правой кнопкой мыши по элементу. Про набор функций, отображаемых в этом меню будет подробнее сказано в пункте 3.1.5.2.

3.1.5.1.6 Панель таблицы

Панель таблицы (рисунок 8) расположена справа от панели дерева

(пункт 3.1.5.1.5) и служит для отображения свойств выбранного в дереве элемента телеметрии. У элементов разного типа набор свойств (пункт 3.1.5.2.9) разный, поэтому вид таблицы различается в зависимости от выбранного элемента. Наборы свойств каждого элемента будут подробнее рассмотрены в пункте 3.1.5.2.

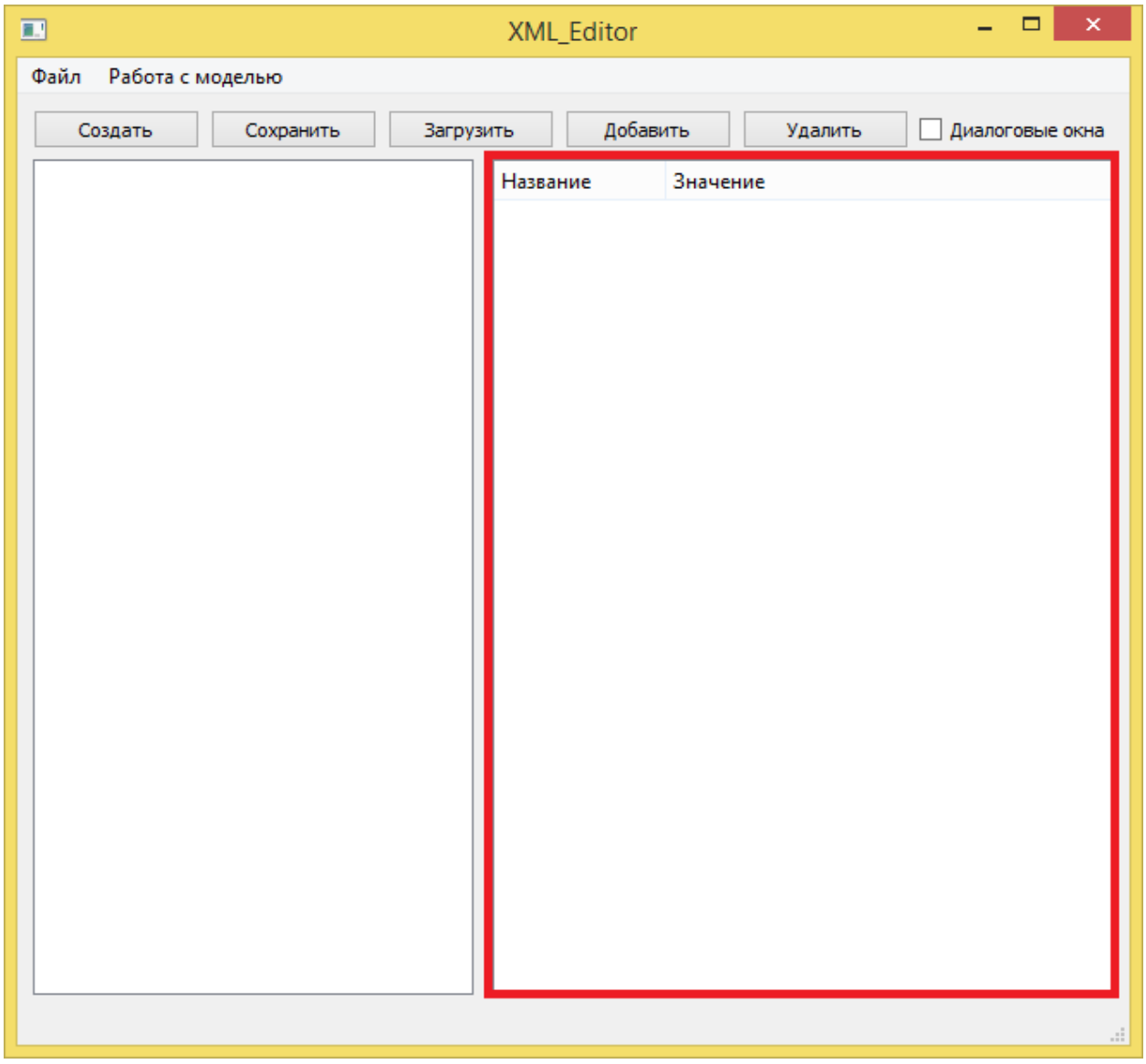


Рисунок 15 – Панель таблицы

3.1.5.1.7 Контекстное меню

При нажатии правой кнопкой мыши на одном из элементов дерева будет выведено контекстное меню, предоставляющее возможность осуществления того или иного действия с данным объектом. В зависимости от того, какой элемент выбран, набор свойств будет включать в себя те или иные операции,

доступные для данного элемента. Пример контекстного меню с максимально доступным набором операций приведён на рисунок 16.

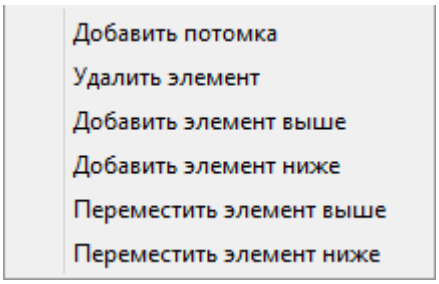


Рисунок 16 - Контекстное меню

Элементы контекстного меню выполняют следующие функции:

- Перемещение выбранного элемента выше или ниже (пункт 3.1.5.2.8)
- Добавление элемента того же типа, что и выбранный (пункт 3.1.5.2.7)
- Добавление дочернего элемента (пункт 3.1.5.2.5)
- Удаление выбранного элемента (пункт 3.1.5.2.6)

3.1.5.2 Описание работы с программой XML-Editor

3.1.5.2.1 Запуск программы

Запуск XML-Editor осуществляется запуском файла «XML-Editor.exe». После успешного запуска программы на экране появится главное окно программы XML-Editor (рисунок 17).

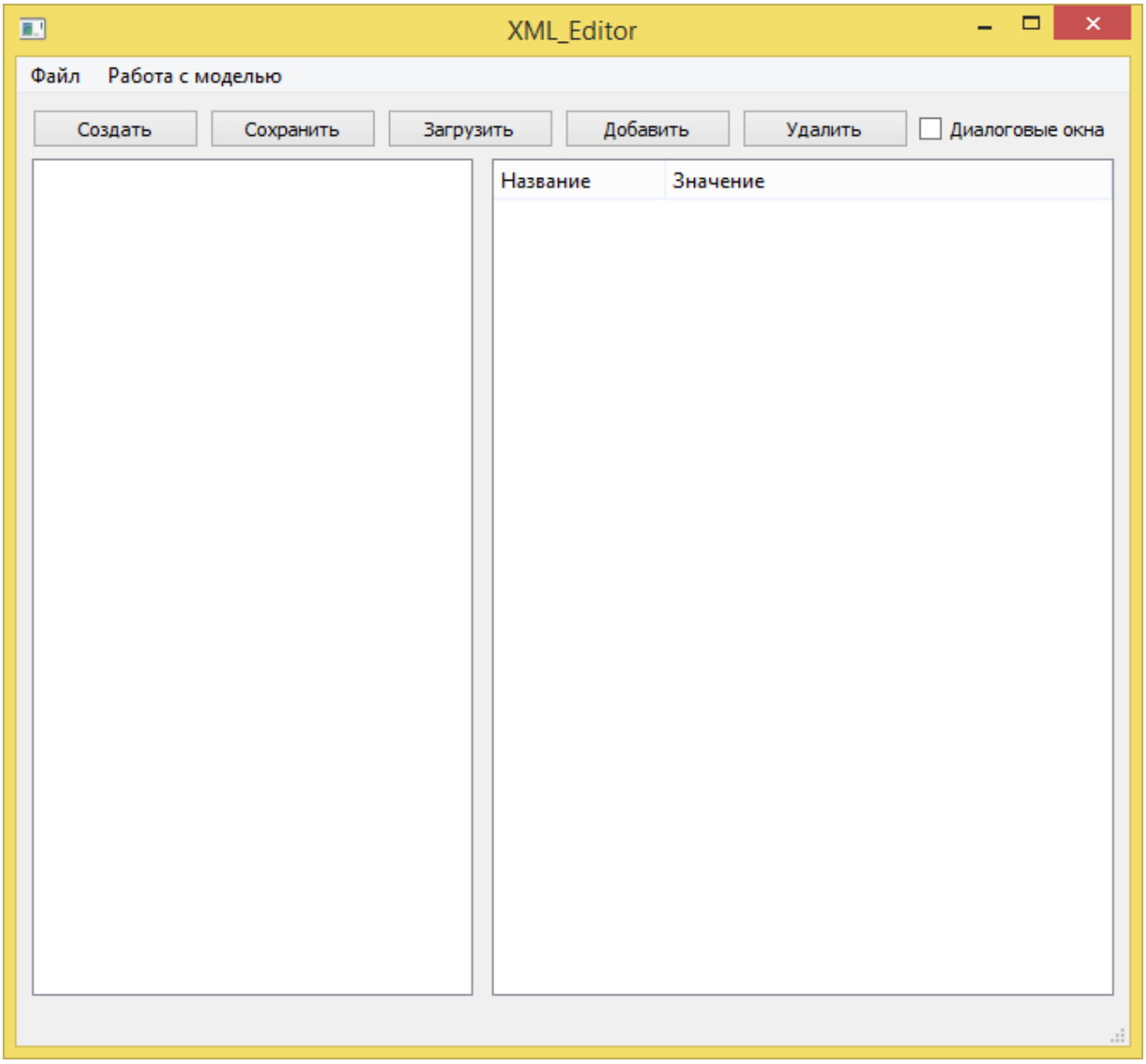


Рисунок 17 - Главное окно программы XML-Editor

Попытка сохранить пустой документ XML не повлияет на работу ПО.

3.1.5.2.2 Создание модели

Чтобы начать работу непосредственно с моделью телеметрии, необходимо выполнить одно из следующих действий:

- создать модель
- загрузить уже существующий XML-документ, моделирующий телеметрию.

Про загрузку модели будет подробно рассказано в пункте 3.1.5.2.3. Для

создания модели необходимо выбрать соответствующий пункт в разделе «Файл» в панели меню (пункт 3.1.5.1.2), либо нажать кнопку «Создать» на панели кнопок быстрого доступа. Результат создания модели приведён на рисунок 18.

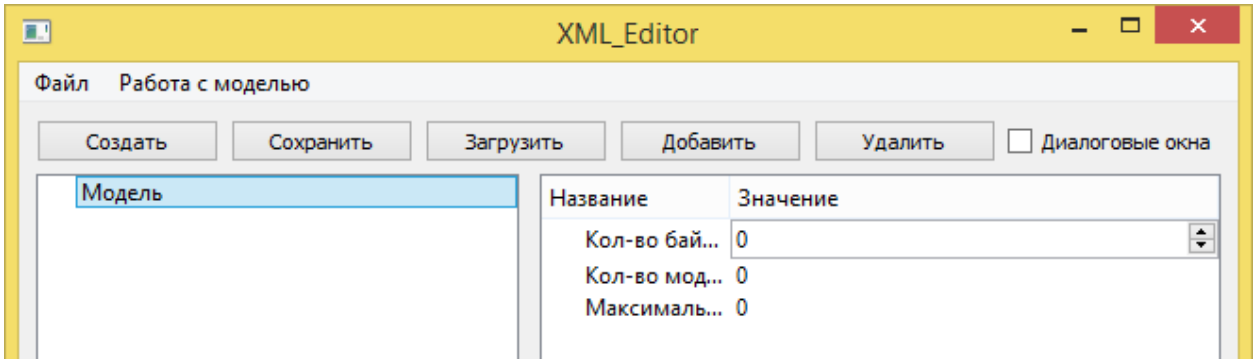


Рисунок 18 - Создание модели

3.1.5.2.3 Загрузка модели

Ещё одним способом перехода к работе с моделью является загрузка уже созданной программно или вручную модели телеметрии. Для этого необходимо выбрать соответствующий пункт в разделе «Файл» в главном меню (пункт 3.1.5.1.2), либо нажать кнопку «Загрузить» на панели кнопок быстрого доступа. В результате выполнения одной из этих операций на экране отобразится окно выбора файла (рисунок 19), после чего нужно выбрать необходимый файл. После выбора файла будет осуществлена проверка формата выбранного XML-документа и в случае успешной проверки в панели дерева отобразится модель выбранного XML-документа (рисунок 20). В случае, если файл не прошёл проверку на формат, на экран будет выведено окно с сообщением об ошибке с описанием того, в каком элементе произошла ошибка и в чём она заключается. В этом случае модель загружена не будет.

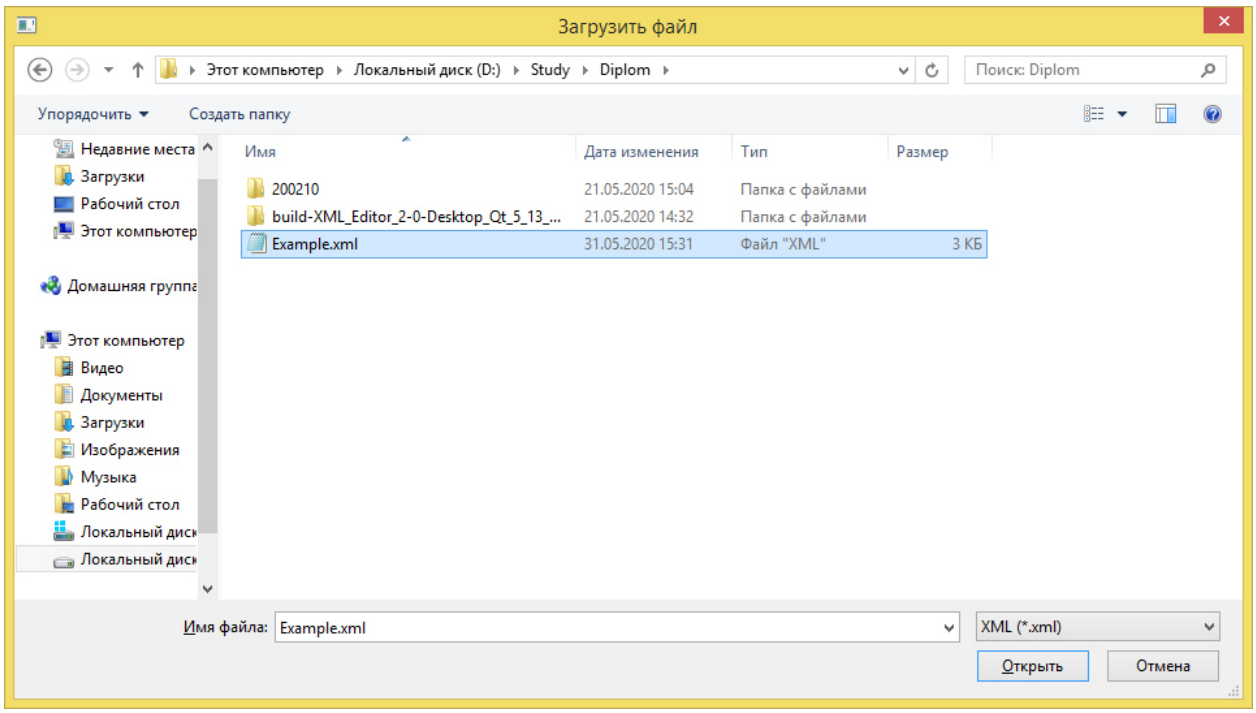


Рисунок 19 - Окно выбора файла

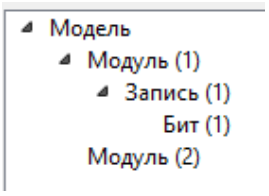


Рисунок 20 - Загруженная модель

3.1.5.2.4 Сохранение модели

Для сохранения модели необходимо выбрать соответствующий пункт в разделе «Файл» в главном меню (пункт 3.1.5.1.2), либо нажать кнопку «Сохранить» на панели кнопок быстрого доступа. В результате выполнения одной из этих операций на экране отобразится окно выбора файла (рисунок 21), в котором необходимо выбрать имя файла. Для сохранения модели необходимо наличие хотя бы корневого элемента XML-документа. В случае сохранения пустого документа на экран будет выведено сообщение об ошибке.

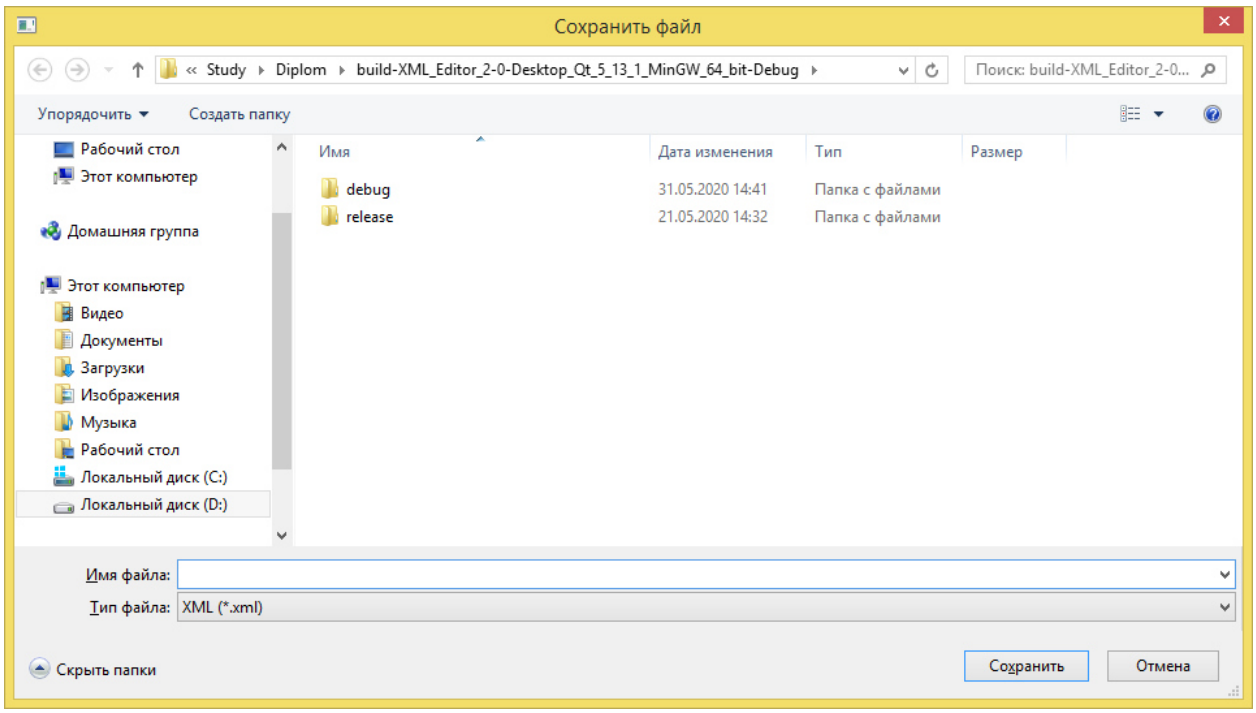


Рисунок 21 - Окно сохранения модели

3.1.5.2.5 Добавление элемента

Для добавления элемента необходимо сначала выбрать родительский элемент в панели дерева (пункт 3.1.5.1.5). Далее можно осуществить добавление следующими способами:

- Кликнуть правой кнопкой мыши по родительскому элементу и в контекстном меню* выбрать пункт: «Добавить потомка».
- Нажать кнопку «Добавить» в панели кнопок быстрого доступа
- Выбрать пункт «Работа с моделью» в панели меню и нажать кнопку «Добавить элемент».

*Примечание – в контекстном меню пункт «Добавить потомка» становится доступным только в случае, если добавление элемента допустимо.

В случае попытки добавить элемент к биту или небитовой записи (Подробнее о свойствах записи в пункте 3.1.5.2.9.3) на экран будет выведено сообщение о невозможности осуществления этого действия.

Соответствие родительских элементов и потомков элементов модели телеметрии представлено в таблице 1.

Таблица 1 – родительские элементы и их потомки

№ п/п	Элемент	Потомок
1	Модель	Модуль
2	Модуль	Запись
3	Запись*	Бит

*Примечание – добавить бит можно только к битовой записи. Подробнее в пункте 3.1.5.2.9.3

3.1.5.2.6 Удаление элемента

Для удаления элемента необходимо сначала выбрать его в панели дерева. Далее можно осуществить добавление следующими способами:

- Кликнуть правой кнопкой мыши по родительскому элементу и в контекстном меню выбрать пункт: «Удалить элемент».
- Нажать кнопку «Удалить»* в панели кнопок быстрого доступа
- Выбрать пункт «Работа с моделью» в панели меню и нажать кнопку «Удалить элемент»*.

*Примечание – удалить можно любой элемент, кроме модели. Удаление элемента подразумевает также удаление всех его потомков. В случае, если удаляемый элемент не выбран, или выбрана модель, нажатие кнопки «Удалить» в панели кнопок быстрого доступа или «Работа с моделью» в панели меню и «Удалить элемент» ни к чему не приведёт.

3.1.5.2.7 Добавление элемента выше или ниже выбранного

Для осуществления этой операции необходимо кликнуть по элементу правой кнопкой мыши и в контекстном меню выбрать соответствующий пункт. В результате выполнения операции будет создан элемент того же уровня вложенности, что и выбранный, и помещён перед или после

выбранного. В программе не может существовать одновременно двух моделей, поэтому добавление второй модели не поддерживается программой. Для всех остальных узлов дерева операция поддерживается вне зависимости от их свойств.

3.1.5.2.8 Перемещение элемента выше или ниже

Чтобы переместить элемент, необходимо кликнуть по нему правой кнопкой мыши и выбрать соответствующий пункт контекстного меню. Данная операция доступна только для тех узлов дерева, которые имеют хотя бы один узел одного уровня с ними. Операция работает независимо от наличия в документе комментариев или узлов, не имеющих отношения к структуре документа. Примеры перемещения элементов приведены на рисунок 22-24

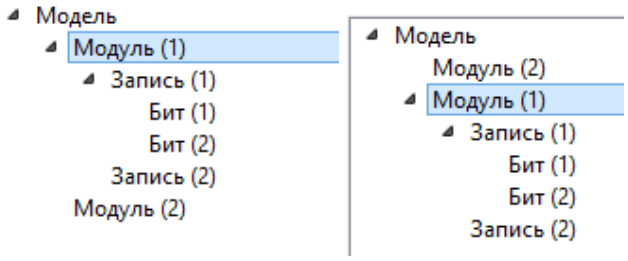


Рисунок 22 - Перемещение модуля

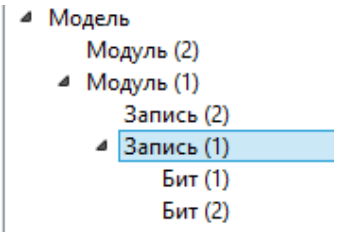


Рисунок 23 - Перемещение записи

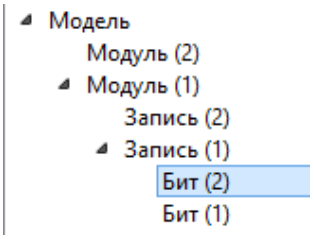


Рисунок 24 - Перемещение бита

3.1.2.5.9 Работа со свойствами

Если в панели дерева (пункт 3.1.5.1.5) выбрать элемент, то в панели таблицы (пункт 3.1.5.1.6) отобразится его набор свойств. Ниже подробно описаны свойства каждого элемента документа, их возможные значения и влияние их изменения на другие элементы документа.

3.1.2.5.9.1 Свойства модели

При выборе модели в панели дерева (пункт 3.1.5.1.5) в панели таблицы (пункт 3.1.5.1.6) отразится набор свойств модели. На рисунке 25 представлен набор свойств модели.

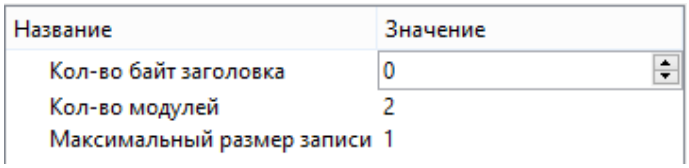


Рисунок 25 - Набор свойств модели

Как видно из рисунка 25, набор свойств модели включает в себя следующие поля:

- Количество байт заголовка – единственное доступное для редактирования поле таблицы. Задаётся пользователем в размере от 0 до 4294967295. Изменение количества байт заголовка не влияет на свойства других элементов.
- Количество модулей – показывает, сколько модулей содержит модель телеметрии. Недоступно для редактирования пользователем, рассчитывается автоматически.
- Максимальный размер записи – отображает максимальный размер записи. Недоступно для редактирования, рассчитывается автоматически.

3.1.2.5.9.2 Свойства модуля

При выборе модуля в панели дерева в панели таблицы отразится набор свойств модуля. На рисунок 26 представлен набор свойств модуля. Все поля доступны для редактирования.

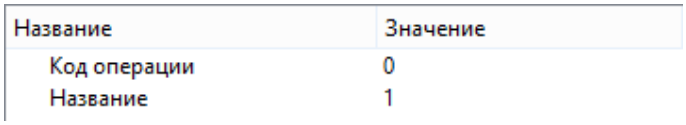


Рисунок 26 - Набор свойств модуля

Как видно из рисунка 26, набор свойств модуля включает в себя следующие поля:

- Код операции – поле, значение которого может быть десятичным или шестнадцатеричным числом в диапазоне от 0 до 255. Формат шестнадцатеричного числа здесь и далее: перед числом обязательно должны стоять символы «0x», регистр букв a-f значения не имеет.
- Название – отображает название модуля. При изменении названия текущего элемента меняется его название и в панели дерева. Всё вышеперечисленное верно также для свойств «Название» и у других элементов.

3.1.2.5.9.3 Свойства записи

При выборе записи в панели дерева в панели таблицы отразится набор свойств записи. На рисунке 27 представлен набор свойств записи. Все поля доступны для редактирования.

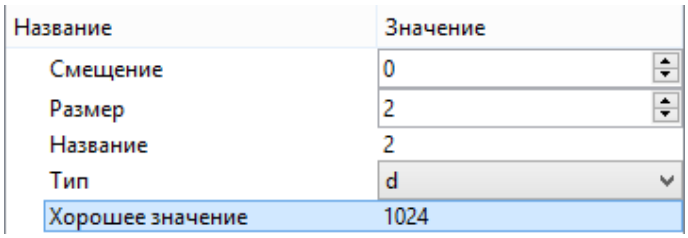


Рисунок 27 - Свойства записи

Как видно из рисунка 27, набор свойств записи включает в себя следующие поля:

- Смещение – поле, имеющее беззнаковый целочисленный формат. Задаётся пользователем.
- Размер - имеющее беззнаковый целочисленный формат. Задаётся пользователем. Размер записи влияет на максимальное допустимое значение поля «хорошее значение» в случае, если тип не битовый и на биты, если тип поля битовый. При изменении значения поля, если установлен флаг отображения вспомогательных диалоговых окон, то после проверки типа поля, «Хорошего значения» или бит, на экран будет выведено сообщение, предупреждающее о том, что изменение размера записи повлияет на «Хорошее значение» или биты. Если будет решено согласится на изменение размера, то хорошее значение будет удалено, или будет проведена проверка размера бит на соответствие новому значению, и биты, которые не соответствуют размеру записи, будут удалены. В противном случае будет восстановлено старое значение размера записи. Если флаг установлен не будет, то вышеперечисленные удаления произойдут без вывода предупреждающего сообщения. Кроме того, если значение этого поля находится вне диапазона [1;4], то добавление поля «Хорошее значение» невозможно.
- Название – поле, аналогичное названию модуля.
- Тип – очень важное поле, отвечающее за возможность добавления «Хорошего значения» или бит. Имеет 5 допустимых значения: “d”, “x”, “b”, “mac”, “ip”. Поле «Хорошее значение» доступно только когда поле «Тип» имеет значение “d” или “x”, а добавление бит – когда поле имеет значение “b”. При изменении значения с “d” или “x” на “b”, “mac” или “ip” произойдёт удаление поля «Хорошее значение», если таковое имеется. При изменении значения с “b” на любое другое все биты будут

удалены. В случае, когда тип - “mac” или “ip”, поле «Размер» принимает константное значение – 6 или 4 соответственно.

- Хорошее значение – поле, которое доступно пользователю только при следующих условиях: тип записи не битовый и размер находится в диапазоне [1;4]. Минимальное значение – 0. Максимальное значение высчитывается в зависимости от размера по формуле: «m =1 << s*8», где s – размер записи, m – максимальное хорошее значение. Может быть записано в десятичном или шестнадцатеричном формате, аналогичном коду операции модуля.

3.1.2.5.9.4 Свойства бита

При выборе бита в панели дерева в панели таблицы отразится набор свойств бита. На рисунке 28 представлен набор свойств бита. Все поля доступны для редактирования. Все поля доступны для редактирования.

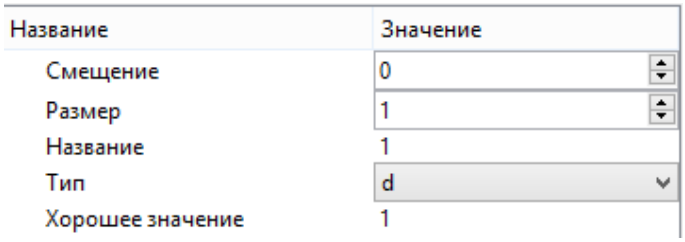


Рисунок 28 - Свойства бита

Как видно из рисунка 28, набор свойств бита включает в себя следующие поля:

- Смещение – поле, имеющее беззнаковый целочисленный формат. Задаётся пользователем. В сумме с размером не должно быть больше размера родительской записи.
- Размер – поле, имеющее беззнаковый целочисленный формат. Задаётся пользователем. В сумме со смещением не должно быть больше размера родительской записи. При значениях в диапазоне [1;32] становится доступным поле «Хорошее значение».
- Название – поле, аналогичное одноимённым полям модуля и записи.

- Тип – Имеет 3 допустимых значения: “d”, “x”, “b”.
- Хорошее значение - поле, которое доступно пользователю только при условии, если размер находится в диапазоне [1;32]. Минимальное значение – 0. Максимальное значение высчитывается в зависимости от размера по формуле: «m =1 << s», где s – размер записи, m – максимальное хорошее значение. Может быть записано в десятичном или шестнадцатеричном формате, аналогичном коду операции модуля и одноимённого поля записи.

3.1.2.5.10 Сообщения об ошибках

При загрузке XML-документа и во время работы с ним могут возникнуть ошибки. Их можно условно подразделить на два типа:

- Ошибки, возникшие при проверке XML-документа
- Ошибки, возникшие по вине пользователя.

3.1.2.5.10.1 Ошибки, возникающие при проверке XML-документа

Примеры ошибок приведены ниже на рисунках 29-45.

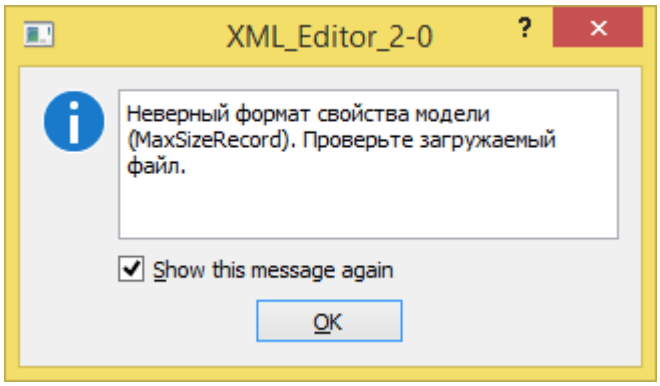


Рисунок 29 – Ошибка в XML-документе – неверное значение свойства модели

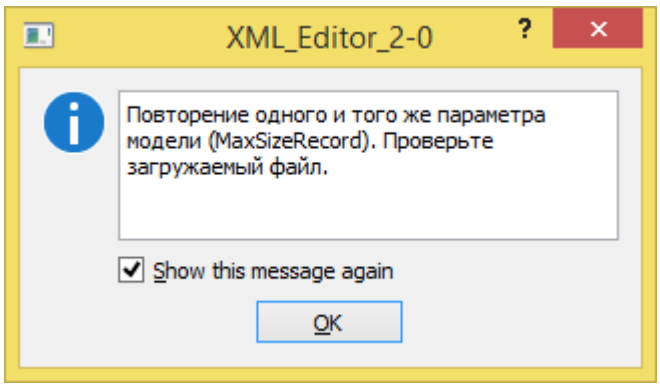


Рисунок 30 - Ошибка в XML-документе – повторение свойства модели

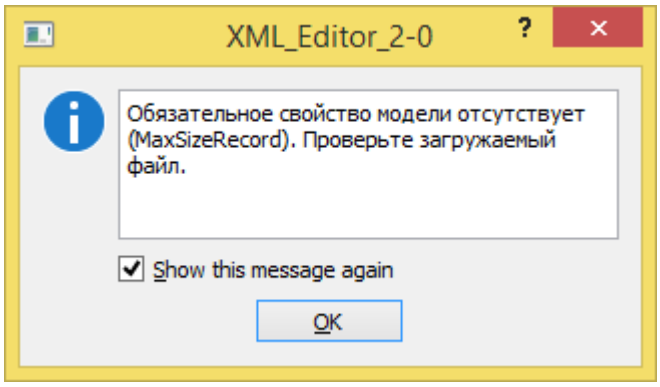


Рисунок 31 - Ошибка в XML-документе – отсутствие свойства модели

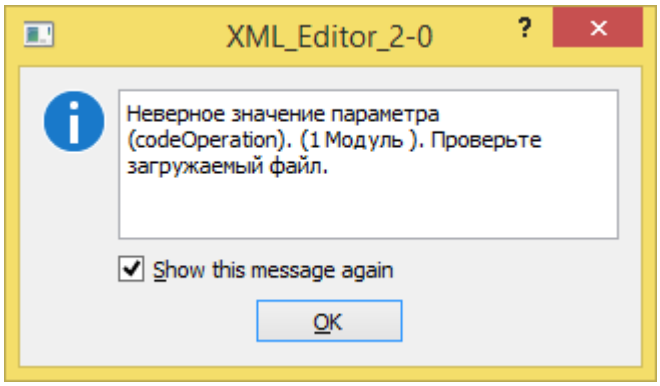


Рисунок 32 - Ошибка в XML-документе – неверное значение свойства модуля

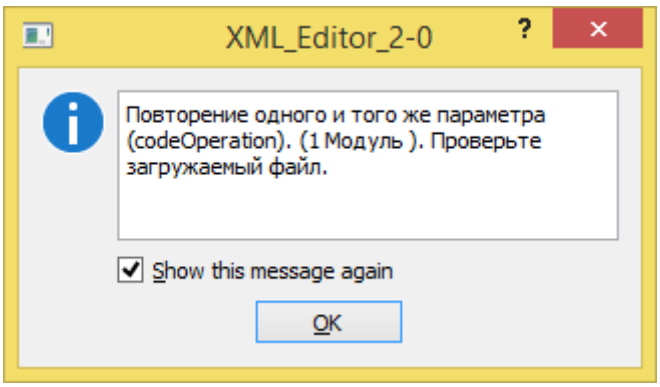


Рисунок 33 - Ошибка в XML-документе – повторение свойства модуля

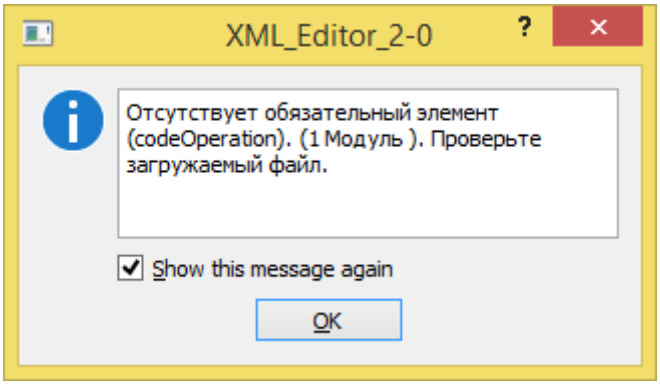


Рисунок 34 - Ошибка в XML-документе – отсутствие свойства модуля

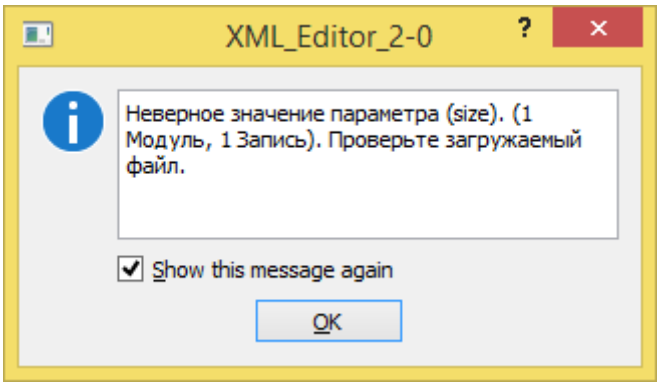


Рисунок 35 - Ошибка в XML-документе – неверный формат свойства блока

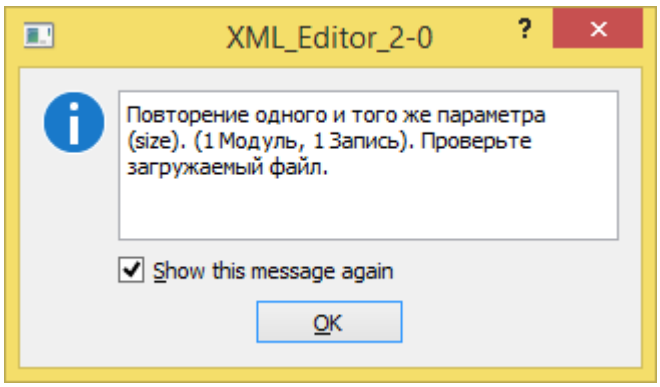


Рисунок 36 - Ошибка в XML-документе – повторение свойства блока

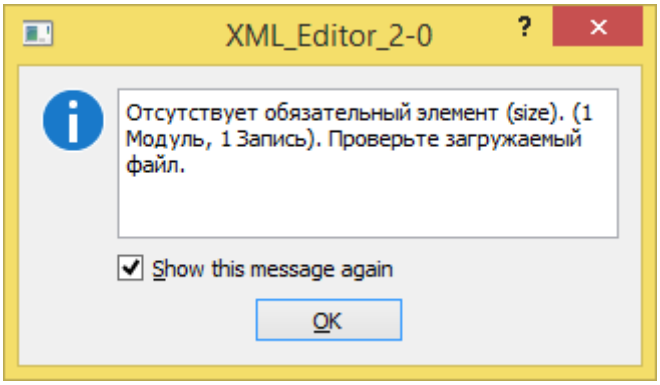


Рисунок 37 - Ошибка в XML-документе – отсутствие обязательного свойства блока

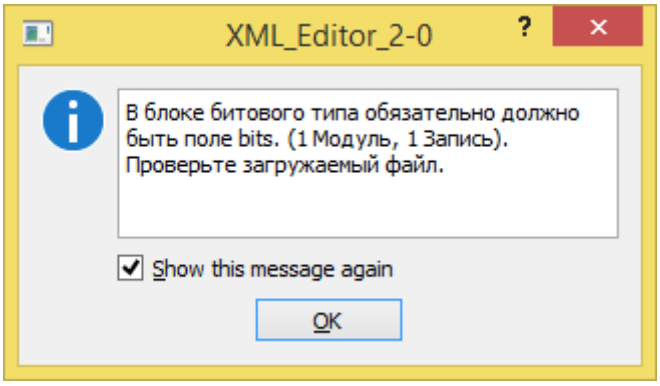


Рисунок 38 - Ошибка в XML-документе – отсутствие обязательного свойства битового блока

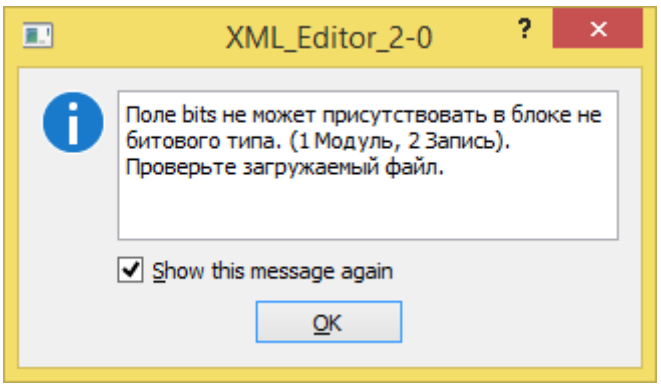


Рисунок 39 – Ошибка в XML-документе – наличие неверного свойства в небитовом блоке

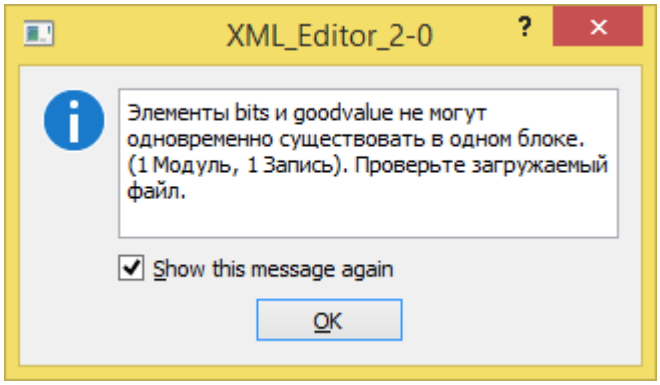


Рисунок 40 – Ошибка в XML-документе – наличие взаимопротиворечащих свойств в одном блоке

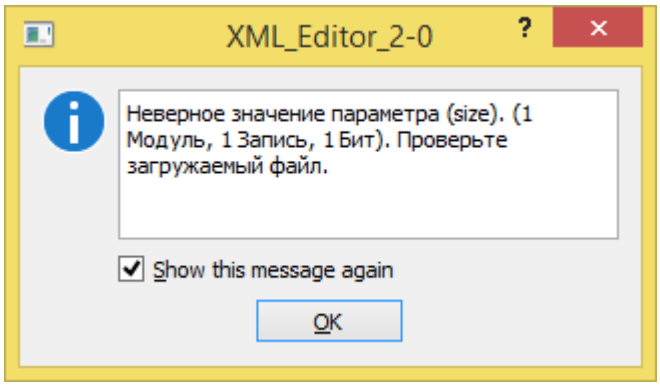


Рисунок 41 - Ошибка в XML-документе – неверный формат свойства бита

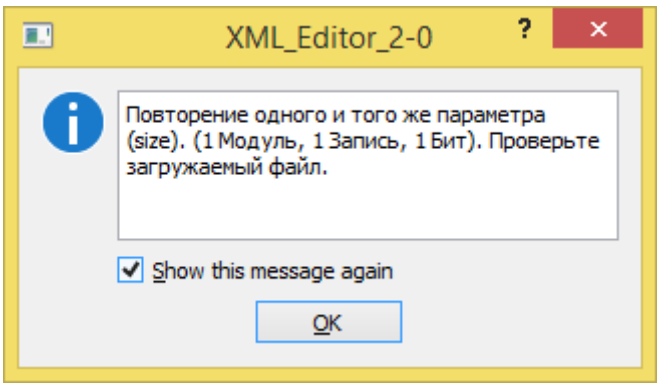


Рисунок 42 - Ошибка в XML-документе – повторение свойства бита

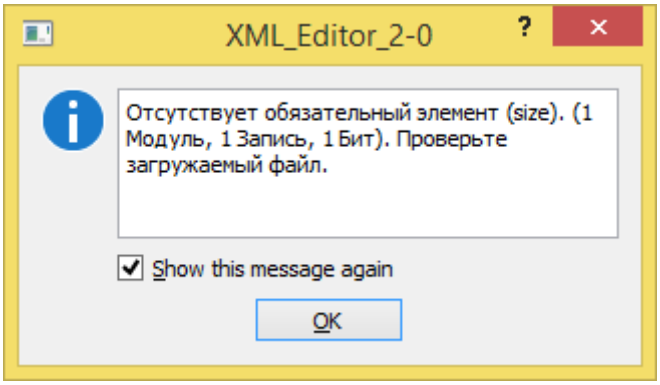


Рисунок 43 Ошибка в XML-документе – отсутствие обязательного свойства бита

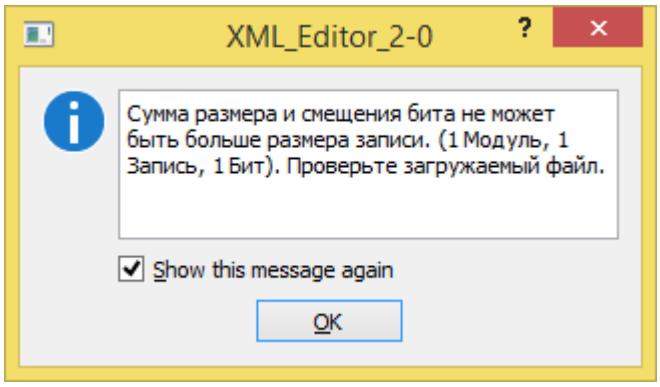


Рисунок 44 - Ошибка в XML-документе – несоответствие суммы размера и смещения бита размеру записи

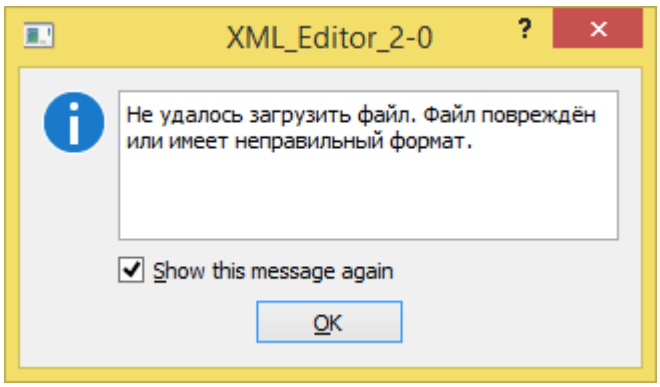


Рисунок 45- Ошибка в XML-документе – другая ошибка.

3.1.2.5.10.2 Ошибки, возникающие из-за действий оператора

Примеры ошибок, возникающих из-за действий оператора приведены на рисунках 46 – 50.

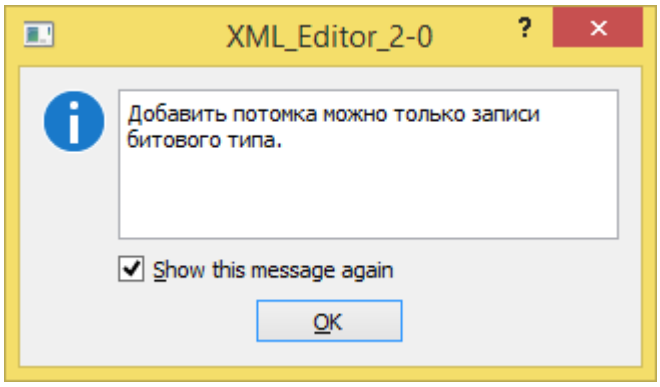


Рисунок 46 – Попытка добавить потомка не битовой записи

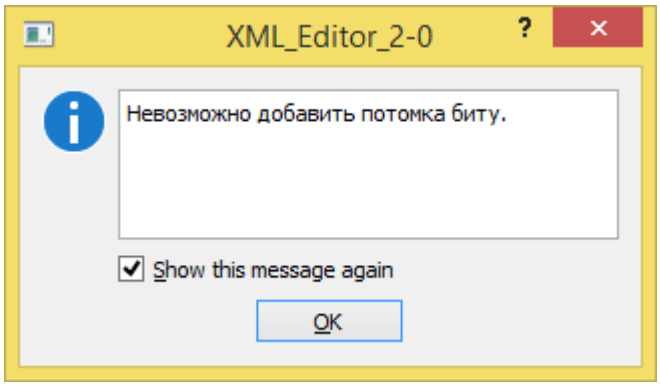


Рисунок 47 – Попытка добавить потомка биту

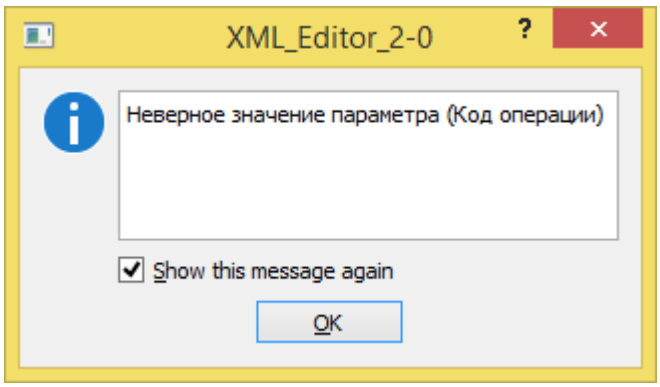


Рисунок 48 – Попытка ввести некорректное значение в поле «код операции»

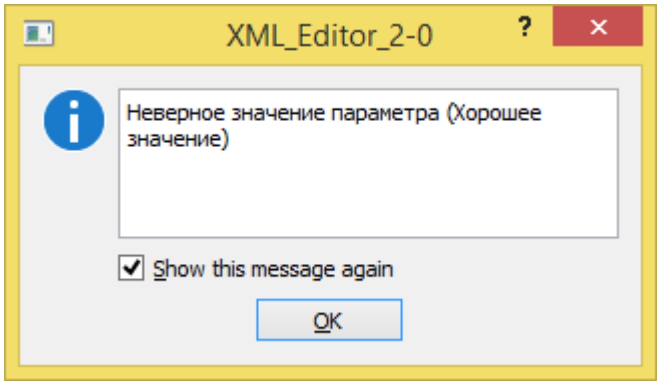


Рисунок 49 – Попытка ввести некорректное значение в поле «хорошее значение»

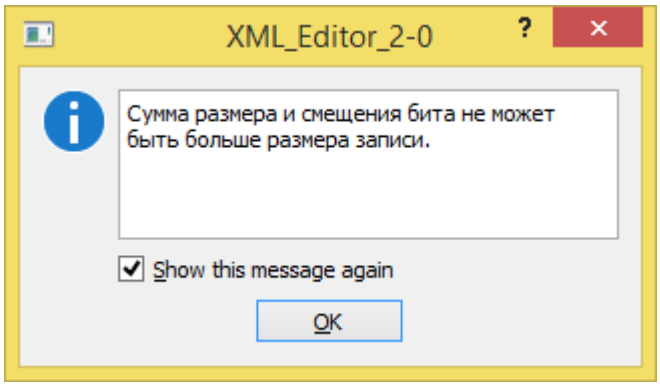


Рисунок 50 – Попытка задать размер или смещение бита таковым, что их сумма будет больше размера блока

3.1.5.2.11 Диалоговые сообщения

Во время работы с программой могут возникнуть ситуации, в которых изменение одного параметра влечёт за собой изменение или удаление

каких-либо параметров других элементов, либо самих элементов. Такие действия требуют от пользователя подтверждения своих намерений. Для этого предусмотрены специальные диалоговые сообщения, которые позволят пользователю подтвердить свои намерения, либо отказаться от них. За вывод на экран данных сообщений отвечает флаг отображения вспомогательных диалоговых окон (пункт 3.1.5.1.4). В случае, если флаг не установлен, все изменения происходят автоматически. Примеры сообщений приведены на рисунках 51- 54.

При изменении размера битовой записи может случиться ситуация, при которой сумма размера и смещения одного или нескольких бит окажется больше нового размера записи. В этом случае на экран будет выведено сообщение (рисунок 51), требующее от пользователя решения по поводу удаления бит. Если пользователь нажмёт «Да», то новый размер записи сохранится, и все биты, не соответствующие ему, будут удалены.

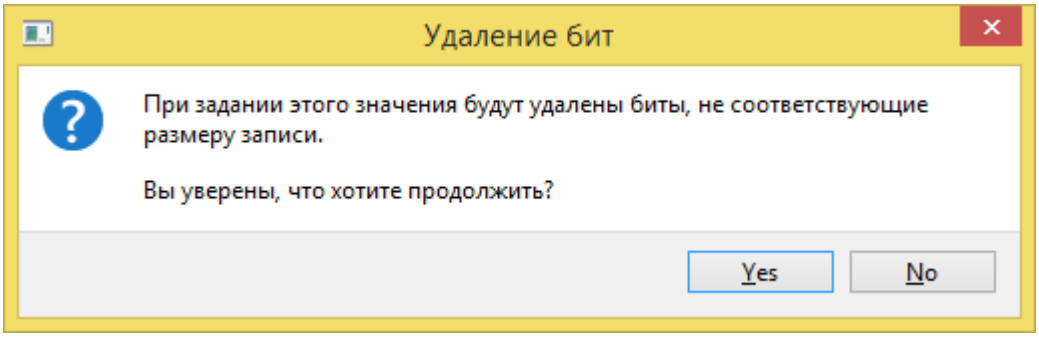


Рисунок 51 – Диалоговое сообщение

При изменении размера не битовой записи может случиться ситуация, когда параметр «Хорошее значение» уже не соответствует новому размеру. В этом случае на экран будет выведено сообщение (рисунок 52), требующее от пользователя подтверждения своих действий. В случае, если пользователь решит всё равно изменить размер записи, поле «Хорошее значение» будет удалено.

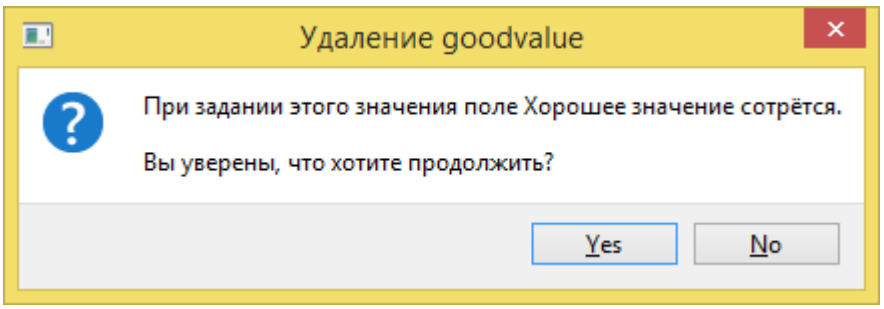


Рисунок 52 - Диалоговое сообщение

При изменении размера или смещения бита может случиться ситуация, когда параметр «Хорошее значение» уже не соответствует сумме размера и смещения бита. В этом случае на экран будет выведено сообщение (рисунок 53), требующее от пользователя подтверждения своих действий. В случае, если пользователь решит всё равно изменить размер или смещение бита, поле «Хорошее значение» будет удалено.

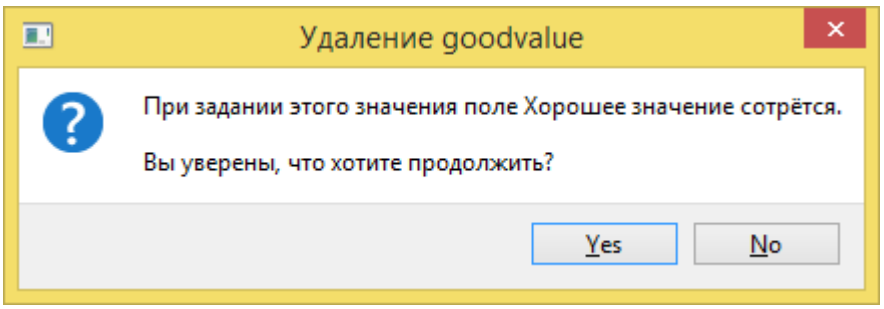


Рисунок 53 - Диалоговое сообщение

При условии, что в программе загружена или создана модель телеметрии, пользователь может создать новую модель. В этом случае старая модель телеметрии будет удалена. Чтобы пользователь мог подтвердить своё намерение создать новую модель, на экран будет выведено сообщение (рисунок 54), запрашивающее от пользователя подтверждение своих намерений.

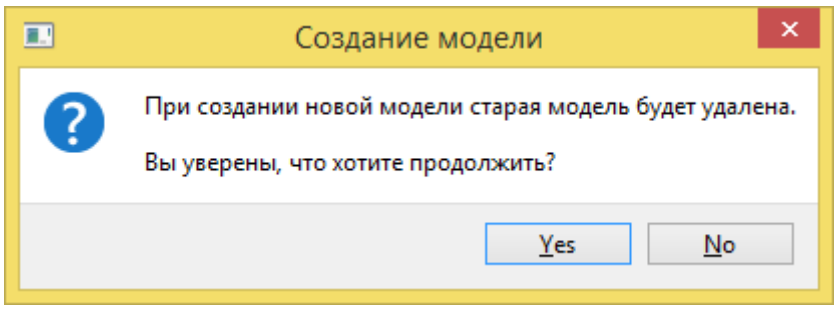


Рисунок 54 - Диалоговое сообщение

3.2 Описание разработки

Методы класса XML_Editor представлены в таблице 2.

Таблица 2 - Методы класса XML_Editor

Название	**Описание**
void on_action_5_triggered()	Функция обработки нажатия на кнопку «Выход»
void on_pushButton_3_clicked()	Функция обработки нажатия на кнопку "Загрузить файл"
void on_treeWidget_currentItemChanged (QTreeWidgetItem *CCurrentTreeItem, QTreeWidgetItem *CPreviousTreeItem)	Функция обработки изменения выбранного элемента дерева CCurrentTreeItem – текущий элемент дерева CPreviousTreeItem – предыдущий элемент дерева
void on_pushButton_2_clicked()	Функция обработки нажатия на кнопку "Сохранить файл"
void on_checkBox_stateChanged (int arg1)	Функция обработки изменения состояния чек-бокса nArg1 – Состояние чек-бокса
void on_pushButton_4_clicked()	Функция обработки нажатия на кнопку "Добавить"
void on_pushButton_clicked()	Функция обработки нажатия на кнопку "Создать"
void on_pushButton_5_clicked()	Функция обработки нажатия на кнопку "Удалить"
void ShowContextMenu(const QPoint& pos)	Функция вывода контекстного меню CPos – Положение вывода
void ReplaceElementHigher()	Функция перемещения элемента выше
void ReplaceElementLower()	Функция перемещения элемента ниже
void AddElementHigher()	Функция добавления элемента выше
void AddElementLower()	Функция добавления элемента ниже
int GetNodeLevel (QTreeWidgetItem *CItem)	Функция получения уровня вложенности выбранного элемента дерева CItem – Элемент дерева Возвращает глубину вложенности элемента

Продолжение таблицы 2

void UpdateTree (QDomDocument* CXDoc, QStringList* aCElementNamesE, QStringList* aCElementNamesR, QList< QStringList > *aaCNamesString)	Функция обновления дерева. CXDoc – XML-документ aCElementNamesE – Массив названий элементов XML aCElementNamesR – Массив названий элементов XML на русском aaCNamesString – Массив названий свойств XML
void UpdateTreeUnitsRecords (QTreeWidgetItem *CCurrentItem, QDomNode CParentXmlNode, bool bIsUnit, QStringList* aCElementNamesE, QStringList* aCElementNamesR, QList< QStringList > *aaCNamesString)	Функция обновления дерева для модулей и блоков. CCurrentItem – Текущий элемент дерева CParentXmlNode – Текущий элемент XML bIsUnit – Признак того, что элемент – модуль aCElementNamesE – Массив названий элементов XML aCElementNamesR – Массив названий элементов XML на русском aaCNamesString – Массив названий свойств XML
void UpdateTreeBits (QTreeWidgetItem *CCurrentItem, QDomNode CXmlBit, QString CXStringName, QString CBitName)	Функция обновления дерева для битов. CCurrentItem – Текущий элемент дерева CXmlBit – Текущий элемент XML CXStringName – Название параметра, в котором находится название бита CBitName – Название бита на русском aCElementNamesR – Массив названий элементов XML на русском aaCNamesString – Массив названий свойств XML
QDomNode FindXml (QDomDocument *CXDoc, QTreeWidgetItem *CSelectedNode, QList< QStringList > *aaCNamesString, QStringList *aCElementNamesE)	Функция поиска XML-элемента по выбранному элементу дерева CXDoc – XML-документ CSelectedNode – текущий элемент дерева aaCNamesString – массив названий свойств XML aCElementNamesE – массив названий элементов XML Возвращает XML-элемент, соответствующий выбранному элементу дерева

Продолжение таблицы 2

QDomNode FindUnit (QDomNode *CCurrentXML, int nCCurrentNodeIndex, QStringList *aCElementNamesE)	Функция поиска XML-элемента типа Unit по выбранному элементу дерева. CCurrentXML – текущий элемент XML nCCurrentNodeIndex – индекс текущего эл-та дерева aCElementNamesE – массив названий элементов XML Возвращает XML-элемент типа Unit
QDomNode FindRecordOrBit (QDomNode *CCurrentXML, int nCCurrentNodeIndex, QString CContainerName, QString CXmlName)	Функция поиска XML-элемента типа record или bit по выбранному элементу дерева CCurrentXML – текущий элемент XML nCCurrentNodeIndex – индекс текущего эл-та дерева CXmlName– название элемента, XML Возвращает XML-элемент типа record или bit
void SetupMenu()	Функция создания контекстного меню
void EndOfCheckTable (bool bNeedUpdateTree, bool bNeedToUpdateMaxSizeRecord)	Функция реагирования на изменение таблицы bNeedUpdateTree – флаг обновления дерева bNeedToUpdateMaxSizeRecord – флаг обновления максимального размера записи

Методы класса XML_Check представлены в таблице 3.

Таблица 3 - Методы класса XML_Check

Название	**Описание**
static bool CheckUnits (QDomElement *CUnits, QList<QStringList> *aCNames, QStringList *aCElementNamesE, QStringList *aCElementNamesR, QStringList aCErrorMessages, QStringList aCTypeVal)	Функция проверки корневого эл-та XML-документа CUnits – корневой эл-т XML-документа aCNames – массив названий свойств XML aCElementNamesE – массив названий элементов XML aCElementNamesR – массив имён параметров на русском языке aCErrorMessages – массив сообщений об ошибке aCTypeVal – массив значений свойства "type" Возвращает признак правильности эл-та
static bool CheckUnits (QDomElement *CUnits, QList<QStringList> *aCNames, QStringList *aCElementNamesE, QStringList *aCElementNamesR, QStringList aCErrorMessages, QStringList aCTypeVal)	Функция проверки корневого эл-та XML-документа CUnits – корневой эл-т XML-документа aCNames – массив названий свойств XML aCElementNamesE – массив названий элементов XML aCElementNamesR – массив имён параметров на русском языке aCErrorMessages – массив сообщений об ошибке aCTypeVal – массив значений свойства "type" Возвращает признак правильности эл-та

Продолжение таблицы 3

static bool CheckUnits (QDomElement *CUnits, QList<QStringList> *aCNames, QStringList *aCElementNamesE, QStringList *aCElementNamesR, QStringList aCErrorMessages, QStringList aCTypeVal)	Функция проверки корневого эл-та XML-документа CUnits – корневой эл-т XML-документа aCNames – массив названий свойств XML aCElementNamesE – массив названий элементов XML aCElementNamesR – массив имён параметров на русском языке aCErrorMessages – массив сообщений об ошибке aCTypeVal – массив значений свойства "type" Возвращает признак правильности эл-та
static bool CheckUnit (QDomNode *CUnit, QList<QStringList> *aCNames, QStringList *aCElementNamesE, QStringList *aCElementNamesR, QStringList aCErrorMessages, QStringList aCTypeVal, int nCurrentModule)	Функция проверки эл-та XML-документа типа Unit CUnit – проверяемый эл-т типа Unit aCNames – массив названий свойств элементов телеметрии aCElementNamesE – массив названий элементов XML aCElementNamesR – массив имён параметров на русском языке aCErrorMessages – массив сообщений об ошибке aCTypeVal – массив значений элемента "type" nCurrentModule – номер модуля по списку Возвращает признак правильности эл-та
static int CheckUnitValue (QDomNode *CUnitParam, int nPos, QStringList *CNames, int nWrongValue)	Функция проверки формата параметров эл-та XML-документа типа Unit CUnitParam – проверяемый параметр эл-та типа Unit nPos – Расположение проверяемого эл-та в списке CNames – массив названий свойств элементов телеметрии nWrongValue – Признак ошибки Возвращает положение эл-та в списке, если его формат верен, или признак неправильности эл-та

Продолжение таблицы 3

static bool IsUint (QString CValue)	Функция проверки строки на соответствие типу uint CValue – проверяемая строка Возвращает True если строка соответствует uint, False если не соответствует
static bool CheckRecords (QDomNode *CRecords, QList<QStringList> *aCNames, QStringList *aCElementNamesE, QStringList *aCElementNamesR, QStringList aCErrorMessages, QStringList aCTypeVal, int nCurrentUnit)	Функция проверки эл-та XML-документа типа records CRecords – проверяемый эл-т типа records aCNames – массив названий свойств элементов телеметрии aCElementNamesE – массив названий элементов XML aCElementNamesR – массив имён параметров на русском языке aCErrorMessages – массив сообщений об ошибке aCTypeVal – массив значений элемента "type" nCurrentUnit – номер модуля по списку Возвращает признак правильности эл-та
static bool CheckRecords (QDomNode *CRecords, QList<QStringList> *aCNames, QStringList *aCElementNamesE, QStringList *aCElementNamesR, QStringList aCErrorMessages, QStringList aCTypeVal, int nCurrentUnit)	Функция проверки эл-та XML-документа типа records CRecords – проверяемый эл-т типа records aCNames – массив названий свойств элементов телеметрии aCElementNamesE – массив названий элементов XML aCElementNamesR – массив имён параметров на русском языке aCErrorMessages – массив сообщений об ошибке aCTypeVal – массив значений элемента "type" nCurrentUnit – номер модуля по списку Возвращает признак правильности эл-та

Продолжение таблицы 3

static bool CheckRecord (QDomNode *CRecord, QList<QStringList> *aCNames, QStringList *aCElementNamesE, QStringList *aCElementNamesR, QStringList aCErrorMessages, QStringList aCTypeVal, int nCurrentUnit, int nCurrentRecord)	Функция проверки эл-та XML-документа типа record CRecord – проверяемый эл-т типа record aCNames – массив названий свойств элементов телеметрии aCElementNamesE – массив названий элементов XML aCElementNamesR – массив имён параметров на русском языке aCErrorMessages – массив сообщений об ошибке aCTypeVal – массив значений элемента "type" nCurrentUnit – номер модуля по списку nCurrentRecord – номер записи по списку Возвращает признак правильности эл-та
static void CreateRecordParamsArray (QDomNode* CRecord, int(&anPlaceArray)[2][6], QStringList aCTypeVal)	Функция заполнения массива значений эл-та XML-документа типа record CRecord – проверяемый эл-т типа record anPlaceArray – массив положений и значений потомков эл-та типа record aCTypeVal – массив значений элемента "type"
static bool CheckRecordErrors (int(&anPlaceArray)[2][6])	Функция проверки формата потомков эл-та XML-документа типа records anPlaceArray – указатель на массив положений и значений потомков эл-та типа record Возвращает признак правильности эл-та типа record

Продолжение таблицы 3

static void ShowRecordErrors (int(&anPlaceArray)[2][6], QStringList *aCNames, int nCurrentUnit, int nCurrentRecord, QStringList * aCElementNamesR, QStringList *aCErrors)	Функция вывода ошибок эл-та XML-эл-та типа record anPlaceArray – массив положений и значений потомков эл-та типа record aCNames – массив имён потомков XML эл-та типа record nCurrentUnit – номер модуля по списку nCurrentRecord – номер записи по списку aCElementNamesR – массив имён параметров на русском языке aCErrorMessages – массив сообщений об ошибке
static bool CheckBitValue (QDomNode *CBit, int(&anPlaceArray)[5], QStringList *aCNames, QStringList *aCElementNamesR, QStringList aCErrorMessages, QStringList aCTypeVal, int nCurrentUnit, int nCurrentRecord, int nCurrentBit, uint uRecordSize, bool bErrorMessageShow)	Функция проверки элемента типа bit на ошибки CBit – проверяемый эл-т типа bit anPlaceArray – потомки эл-та типа bit aCNames – массив имён св-в XML эл-та типа record aCElementNamesR – массив имён св-в на русском языке aCErrorMessages – массив сообщений об ошибке aCTypeVal –значения элемента "type" nCurrentUnit – номер модуля по списку nCurrentRecord – номер записи по списку nCurrentBit – номер бита по списку uRecordSize – размер записи bErrorMessageShow - флаг, вывода ошибок Возвращает Признак правильности эл-та типа bit
static void ClearUnits (QDomNode CUnits, QStringList *aCElementNamesE, QList<QStringList> *aaCNamesString)	Функция очистки корневого элемента XML-документа CUnits – корневой элемент XML-документа aCElementNamesE – названия элементов телеметрии aaCNamesString – массив названий свойств XML-элементов

Продолжение таблицы 3

static void ClearUnit (QDomNode CUnit, QStringList *aCElementNamesE, QList<QStringList> *aaCNamesString)	Функция очистки элемента Unit CUnit – элемент Unit aCElementNamesE – названия элементов телеметрии aaCNamesString – массив названий свойств XML-элементов
static void ClearRecords (QDomNode CRecords, QStringList *aCElementNamesE, QList<QStringList> *aaCNamesString)	Функция очистки элемента Records CRecords – элемент Records aCElementNamesE – названия элементов телеметрии aaCNamesString – массив названий свойств XML-элементов
static void ClearRecord (QDomNode CRecord, QStringList *aCElementNamesE, QList<QStringList> *aaCNamesString)	Функция очистки элемента Record CRecord – элемент Record aCElementNamesE – названия элементов телеметрии aaCNamesString – массив названий свойств XML-элементов
static void ClearBits (QDomNode CBits, QStringList *aCElementNamesE, QList<QStringList> *aaCNamesString)	Функция очистки элемента Bits CBits – элемент Bits aCElementNamesE – названия элементов телеметрии aaCNamesString – массив названий свойств XML-элементов
static void ClearBits (QDomNode CBits, QStringList *aCElementNamesE, QList<QStringList> *aaCNamesString)	Функция очистки элемента Bits CBits – элемент Bits aCElementNamesE – названия элементов телеметрии aaCNamesString – массив названий свойств XML-элементов
static void ClearBit (QDomNode CBit, QList<QStringList> *aaCNamesString)	Функция очистки элемента Bit CBit – элемент Bit aaCNamesString – массив названий свойств XML-элементов

Продолжение таблицы 3

static bool CheckQDocument (QDomDocument *CQDoc, QList<QStringList> *aCNames, QStringList *aCElementNamesE, QStringList *aCElementNamesR, QStringList aCErrorMessages, QStringList aCTypeVal)	Функция проверки XML-документа на ошибки CQDoc – проверяемый документ aCNames – массив имён потомков XML эл-та типа record aCElementNamesE – массив имён элементов на английском языке aCElementNamesR – массив имён параметров на русском языке aCErrorMessages – массив сообщений об ошибке aCTypeVal – массив значений элемента "type" Возвращает признак правильности документа
static bool CheckBits(QDomNode *CBits, QList<QStringList> *aCNames, QStringList *aCElementNamesE, QStringList *aCElementNamesR, QStringList aCErrorMessages, QStringList aCTypeVal, int nCurrentUnit, int nCurrentRecord, uint uRecordSize, bool bErrorMessageShow)	Функция проверки потомков эл-та типа bits CBits – проверяемый эл-т типа bits aCNames – массив названий свойств элементов телеметрии aCElementNamesE – массив названий элементов XML aCElementNamesR – массив имён параметров на русском языке aCErrorMessages – массив сообщений об ошибке aCTypeVal – массив значений элемента "type" nCurrentUnit – номер модуля по списку nCurrentRecord – номер записи по списку uRecordSize – размер записи (используется в функции проверки потомков эл-та bit) bErrorMessageShow - флаг, влияющий на вывод сообщений об ошибке Возвращает признак правильности эл-та типа bits
static bool CheckBit(QDomNode *CBit, QList<QStringList> *aCNames, QStringList *aCElementNamesR, QStringList aCErrorMessages, QStringList aCTypeVal, int nCurrentUnit, int nCurrentRecord, int nCurrentBit, uint uRecordSize, bool bErrorMessageShow)	Функция проверки эл-та типа bit CBit – проверяемый эл-т типа bit Остальные параметры аналогичны функции «CheckBits». Возвращает признак правильности эл-та типа bit

Продолжение таблицы 3

static bool ValueIsValid (QString CValue, int32_t nP)	Функция проверки значения эл-тов XML-документа, которые могут быть записаны как в десятичном, так и в шестнадцатеричном формате CValue – Значение параметра XML-документа nP – Признак, используемый для вычисления максимально допустимого значения Возвращает признак правильности значения эл-та
static QDomNode FindChildXml (QString CName, QDomNode CParentXML)	Функция поиска свойства элемента в XML CName – имя искомого свойства CParentXml – родительский элемент Возвращает ссылку на элемент
static void CheckUnitsCount (QDomDocument *CXDoc, QList<QStringList> *aCNames, QStringList *acElementNamesE)	Функция обновления параметра CountUnit CXDoc – XML-документ aCNames – массив названий свойств элементов телеметрии acElementNamesE – массив названий элементов телеметрии
static void CalcMaxSizeRecord (QDomDocument *CXDoc, QList<QStringList> *aCNames, QStringList *acElementNamesE)	Функция перемещения расчёта максимального размера поля телеметрии CXDoc – XML-документ aCNames – массив названий свойств элементов телеметрии aCElementNames – названия элементов телеметрии
static void CalcMaxSizeRecord (QDomDocument *CXDoc, QList<QStringList> *aCNames, QStringList *acElementNamesE)	Функция перемещения рассчёта максимального размера поля телеметрии CXDoc – XML-документ aCNames – массив названий свойств элементов телеметрии aCElementNames – названия элементов телеметрии
static QDomNode CreateModel (QDomDocument *CXDoc, QList<QStringList> *aCNames, QStringList *acElementNamesE)	Функция создания модели CXDoc – XML-документ aCNames – массив названий свойств элементов телеметрии acElementNamesE – названия элементов телеметрии Возвращает XML-элемент модели

Продолжение таблицы 3

static QDomNode CreateUnit (QDomDocument *CXDoc, QList<QStringList> *aCNames, QStringList *acElementNamesE)	Функция создания модуля CXDoc – XML-документ aCNames – массив названий свойств элементов телеметрии acElementNamesE – названия элементов телеметрии Возвращает XML-элемент модуля
static QDomNode CreateRecord (QDomDocument *CXDoc, QList<QStringList> *aCNames, QStringList *acElementNamesE, QStringList *aCTypeVal)	Функция создания записи CXDoc – XML-документ aCNames – массив названий свойств элементов телеметрии acElementNamesE – названия элементов телеметрии aCTypeVal – массив значений элемента "type" Возвращает XML-элемент записи
static QDomNode CreateBit (QDomDocument *CXDoc, QList<QStringList> *aCNames, QStringList *acElementNamesE, QStringList *aCTypeVal)	Функция создания бита CXDoc – XML-документ aCNames – массив названий свойств элементов телеметрии acElementNamesE – названия элементов телеметрии aCTypeVal – массив значений элемента "type" Возвращает XML-элемент бита
static QList<QDomNode> *CreateParamList (QList<QDomNode> *aCValues, =QDomNode CParentXmlNode, =QString CSNodeName, =QList<QStringList> *aaCNamesString, =QStringList *aCElementNamesE)	Функция создания списка XML aCValues – Список потомков XML-элемента CParentXmlNode – Текущий XML-элемент CSNodeName – название текущего XML-элемента aaCNamesString – массив названий свойств XML-элементов aCElementNamesE – названия элементов телеметрии Возвращает ссылку на список элементов XML
static void ClearDocument (QDomDocument* CXDoc, QStringList *aCElementNamesE, QList<QStringList> *aaCNamesString)	Функция очистки документа CXDoc – XML-документ aCElementNamesE – названия элементов телеметрии aaCNamesString – массив названий свойств XML-элементов

Методы класса TableTree представлены в таблице 4.

Таблица 4 - Методы класса TableTree

Название	Описание
`void UpdateTable(QList<QDomNode> *aCXmlValues, QString CSNodeName, QStringList *aCElementNamesE)`	Функция обновления таблицы aCXmlValues – список параметров XML-элемента CSNodeName – Название элемента XML acElementNamesE – названия элементов телеметрии
`void ClearTable()`	Функция, очищающая таблицу
`void checkIsEnd ( bool bNeedToUpdateTree, bool bNeedToUpdateMaxSizeRecord )`	Сигнал реагирования на изменение таблицы bNeedUpdateTree – текущий элемент дерева bNeedToUpdateMaxSizeRecord – предыдущий элемент дерева
`void SetValueInZeroRow ( const QString& CValue )`	Функция установки значения параметра в нулевую строку таблицы CValue – Значение
`void SetValueInOneRow ( const QString& CValue )`	Функция установки значения параметра в первую строку таблицы CValue – Значение
`void SetValueInThreeRow ( const QString& CValue )`	Функция установки значения параметра в третью строку таблицы CValue – Значение
`void itemChanged(QTreeWidgetItem *item, int column)`	Функция, обрабатывающая изменение таблицы CItem – Строка таблицы nColumn – Номер столбца
`void LoadTableValues ( QList<QDomNode> *aCXList, QList<QTreeWidgetItem *> *aCItems, int nNodeLevel )`	Функция загрузки значений таблицы aCXList – список параметров XML-элемента aCItems – названия свойств элементов телеметрии nNodeLevel – Уровень вложенности текущего элемента

Безопасность жизнедеятельности

Безопасность жизнедеятельности (БЖД) — наука о комфортном и безопасном взаимодействии человека с техносферой, представляет собой область научных знаний, изучающая опасности угрожающие человеку и разрабатывающие способы защиты от них в любых условиях обитания человека.

Задачи БЖД:

- идентификация опасности распознание и количественная оценка негативных воздействий среды обитания;
- предупреждение воздействия тех или иных негативных факторов на человека;
- защита от опасности;
- ликвидация отрицательных последствий воздействия опасных и вредных факторов;
- создание нормального, то есть комфортного состояния среды обитания человека.

На Земле нет такого человека, которому не угрожают опасности. Реализуясь в пространстве и времени, опасности угрожают не только человеку, но и обществу, государству и в целом всему миру. Поэтому профилактика безопасности и защита от них — актуальнейшая проблема, в решении которой должны быть заинтересованы не только отдельные личности, но и государство, и все мировое сообщество.

В то же время нельзя обеспечить абсолютную безопасность для личности, общества, государства. Под безопасностью понимается такой уровень опасности, с которым на данном этапе развития человечества можно смириться. Безопасность — это приемлемый риск. Чтобы его достичь, необходима выработка идеологии безопасности — формирования

соответствующего уровня мышления и поведения человека и общества в целом. Именно этими проблемами и занимается наука безопасность жизнедеятельности.

Сегодня БЖД опирается на осознанную потребность общества, на правила безопасного повеления, выработанные практикой или смежными областями науки, на законы государства и международного права по безопасности и защите населения. Однако этого недостаточно. В основе БЖД должны лежать систематизированные и обобщенные знания об объективных закономерностях существования и развития природы, человека и общества.

Заключение

В данной работе было разработано приложение для редактирования XML-документа, описывающего телеметрию. Приложение обладает простым и интуитивно понятным пользовательским интерфейсом и реализовывает функционал загрузки, проверки и редактирования XML-документа, представляющего описание телеметрии. Разработка осуществлялась с использованием среды QT Designer и языка C++. В дальнейшем приложение можно улучшить, добавив возможность смены темы, автоматического исправления ошибок.

В ходе выполнения работы, были получены навыки разработки клиент-серверного приложения. Улучшены навыки разработки в среде QT Designer. Получен дополнительный опыт и расширены знания по языку программирования C++.

Список используемых источников

1. https://doc.qt.io/ - документация QT
2. Шлее М. «QT 5.10. Профессиональное программирование на C++».
3. Страуструп Б. «Язык программирования C++».
4. Макконнелл С. «Совершенный код».

Приложение А

Листинги программы

Приложение А. 1 – файл «global_vars.h»

```
/*_____________________________________________ global_vars.h
_____________________________________________ */

/**
    \file       global_vars.h
    \brief      Редактор XML
    \author     Прокуров Д.А.
    \version    1.0
    \date       11.02.2020
    \note       Класс содержит глобальные константы и переменные
для функционирования программы

*/

/*
_____________________________________________________________________________
_____________________________ */

#ifndef GLOBAL_VARS_H
#define GLOBAL_VARS_H

/*
_____________________________________________________________________________
__________ */

#include <QtWidgets>
#include <QDomDocument>

/*
_____________________________________________________________________________
__________ */

// Перечисление элементов дерева (см. g_aCElementNamesE и
g_aCElementNamesR)
enum g_aElements
{

    nEModel = 0,
    nEUnit,
    nERecord,
    nEBit
};

// Коды ошибок при проверке свойств элементов загружаемого
документа
enum g_aXmlParams
{
```

```
    nXPEmptyParametr = -1, // Признак отсутствия элемента
    nXPRepeatParametr = -2, // Признак повторения элемента
    nXPWrongValueParametr = -3, // Признак неверности значения
элемента
};

// Коды значений свойств "offset", "string" и "bits"
enum g_anRecordOffsetStringBitsVal
{
    nROVBitsNull = -1, // Значение свойства "bits": свойство
отсутствует
    nROVRight, // Правильное значение свойства "offset" "string"
или "bits" есть
};

// Коды значений свойства "size"
enum g_anRecordSizeVal
{

    nRSVSize_0_5byte = 0,// Значение свойства "size", равное 0
или 5 и более, но не 6
    nRSVSize_1byte, // Значение свойства "size", равное 1
    nRSVSize_2byte, // Значение свойства "size", равное 2
    nRSVSize_3byte, // Значение свойства "size", равное 3
    nRSVSize_4byte, // Значение свойства "size", равное 4
    nRSVSize_6byte = 6, // Значение свойства "size", равное 6
};

// Коды значений свойства "type"
enum g_anRecordTypeVal
{

    nRTVType_d_x = 0, // Значение свойства "type": d или x
    nRTVType_b, // Значение свойства "type": b
    nRTVType_Mac, // Значение свойства "type": mac
    nRTVType_IP // Значение свойства "type": ip
};

// Коды значений свойства "goodvalue"
enum g_anRecordGoodvalueVal
{

    nRGVGoodvalueWrong = -1, // Значение свойства "goodvalue":
свойство имеет неправильное значение
    nRGVRecordGoodvalueNotFind, // Значение свойства
"goodvalue": свойство отсутствует
    nRGVRecordGoodvalue_8bit, // Значение свойства "goodvalue":
1
    nRGVRecordGoodvalue_16bit, // Значение свойства "goodvalue":
2
    nRGVRecordGoodvalue_24bit, // Значение свойства "goodvalue":
3
```

```
    nRGVRecordGoodvalue_32bit, // Значение свойства "goodvalue":
4
};

// Перечисление свойств модели (см. g_aCModelParamNames)
enum g_aModelParams
{
    nMPCountByteHeader = 0,
    nMPCountUnit,
    nMPMaxSizeRecord
};

// Перечисление свойств модуля (см. g_aCUnitParamNames)
enum g_aUnitParams
{
    nUPcodeOperation = 0,
    nUPinformation,
    nUPrecords
};

// Перечисление свойств записи (см. g_aCRecordParamNames)
enum g_aRecordParams
{
    nRPoffset = 0,
    nRPsize,
    nRPstring,
    nRPtype,
    nRPbits,
    nRPgoodvalue
};

// Перечисление свойств бита (см. g_aCBitParamNames)
enum g_aBitParams
{
    nBPoffset = 0,
    nBPsize,
    nBPstring,
    nBPtype,
    nBPgoodvalue
};

// Перечисление значений свойства "тип" (см. g_aCTypeVal)
enum g_aValTypes
{

    VType_D = 0,
    VType_X,
    VType_B,
    VType_MAC,
    VType_IP
};
```

```
// Перечисление свойств, отображаемых в таблице (см.
g_aCParamRusNames)
enum g_aRusParams
{

    RPCountByteHeader = 0,
    RPCountUnit,
    RPMaxSizeRecord,
    RPcodeOperation,
    RPoffset,
    RPsize,
    RPstring,
    RPtype,
    RPgoodvalue
};

// Индексы массива сообщений об ошибке
enum m_Errors
{

    EMError = 0,
    EMLoadFormatError,
    EMLoadNameError,
    EMLoadFileFailure,
    EMCheckFile,

    EMModelPropFormatError,
    EMModelPropRepeatError,
    EMModelPropNullError,

    EMModulePropFormatError,
    EMCModulePropRepeatError,
    EMCModulePropNullError,

    EMRecordBitsGoodvalueError,
    EMRecordGoodvalueError,
    EMRecordBitsTypeError,
    EMRecordBitsNullError,
    EMRecordMacSizeError,
    EMRecordIpSizeError,

    EMBitOffsetSizeError,

    EMUserAddToBit,
    EMUserAddToNonBitRecord,

    EMTableNullValue,

    EMGoodvalueInWrongType,

    EMSaveFileError
};
```

```
extern QDomDocument *g_CXDoc; // Текущий документ XML
extern QTreeWidgetItem *g_CSelectedTreeItem; // Выбранный
элемент дерева
extern QDomNode g_CCurrentParentNode; // Текущий элемент XML
extern QList<QDomNode> *g_aCXmlParams; // Список свойств
текущего элемента
extern bool g_bAllowDialogWindow; // Признак разрешения
использования диалоговых сообщений

extern const uint g_nMaxByteSize; // Максимальный размер байта
extern const uint g_nMaxBitSize; // Максимальный размер бита
extern QStringList g_aCElementNamesE; // Список названий
элементов XML
extern QStringList g_aCElementNamesR; // Список названий
элементов дерева
extern QStringList g_aCModelParamNames; // Список свойств модели
extern QStringList g_aCUnitParamNames; // Список свойств модуля
extern QStringList g_aCRecordParamNames; // Список свойств
записи
extern QStringList g_aCBitParamNames; // Список свойств бита
extern QList<QStringList> g_aaCNamesString; // Общий список всех
свойств
extern QStringList g_aCParamRusNames; // Названия отображаемых
свойств на русском языке
extern QStringList g_aCTypeVal; // Список значений свойства
"тип"
extern QStringList g_aCErrorMessages; // Список возможных ошибок

class global_vars
{
public:
    global_vars();

};

#endif // GLOBAL_VARS_H
```

Приложение А. 2 – файл «tabletree.h»

```
#ifndef TABLETREE_H
#define TABLETREE_H

#include <QtWidgets>
#include "global_vars.h"
#include "xml_check.h"

QT_BEGIN_NAMESPACE
namespace Ui { class TableTree; }
QT_END_NAMESPACE

class TableTree : public QTreeWidget
{
```

```
    Q_OBJECT

public:
    TableTree();
    void UpdateTable(QList<QDomNode> *aCXmlValues, QString
CSNodeName, QStringList *aCElementNamesE);
    void ClearTable();
    QList<QTreeWidgetItem *> *m_aCModelItems;
    QList<QTreeWidgetItem *> *m_aCUnitItems;
    QList<QTreeWidgetItem *> *m_aCRecordItems;
    QList<QTreeWidgetItem *> *m_aCBitItems;
Q_SIGNALS:
    void checkIsEnd(bool bNeedToUpdateTree, bool
bNeedToUpdateMaxSizeRecord);
public Q_SLOTS:
    void SetValueInZeroRow(const QString& value);
    void SetValueInOneRow(const QString& value);
    void SetValueInThreeRow(const QString& value);
    void itemChanged(QTreeWidgetItem *item, int column);



private:
    enum m_Items
    {
        ICountByteHeader = 0,
        ICountUnit,
        IMaxSizeRecord,

        ICodeOperation,
        IInformation,

        IROffset,
        IRSize,
        IRString,
        IRType,
        IRGoodValue,

        IBOffset,
        IBSize,
        IBString,
        IBType,
        IBGoodValue
    };

    QList<QSpinBox*> *m_aCSpinBoxes;
    QList<QComboBox*> *m_aCComboboxes;
    void LoadTableValues(QList<QDomNode> *aCXList,
QList<QTreeWidgetItem *> *aCItems, int nNodeLevel);
};

#endif // TABLETREE_H
```

Приложение А. 3 – файл «xml_check.h»

```
/*_____________________________________________ xml_check.h
_____________________________________________ */

/**
    \file       xml_check.h
    \brief      Редактор XML
    \author     Прокуров Д.А.
    \version    1.0
    \date       11.02.2020
    \note       Класс содержит методы обработки и программного
взаимодействия с документом XML

*/

/*
_____________________________________________________________________________
_____________________________ */

#ifndef XML_CHECK_H
#define XML_CHECK_H

#include<QDomDocument>
#include"global_vars.h"

class XML_Check
{
private:

    static bool CheckUnits(QDomElement *CUnits,
                           QList<QStringList> *aCNames,
                           QStringList *aCElementNamesE,
                           QStringList *aCElementNamesR,
                           QStringList aCErrorMessages,
                           QStringList aCTypeVal);


    static int CheckUnitsParam(QDomNode *CParam,
                               int nPosition,
                               int nWrongValueFlag);


    static bool CheckUnit(QDomNode *CUnit,
                          QList<QStringList> *aCNames,
                          QStringList *aCElementNamesE,
                          QStringList *aCElementNamesR,
                          QStringList aCErrorMessages,
                          QStringList aCTypeVal,
                          int nCurrentModule);
```

```
    static int CheckUnitValue(QDomNode *CUnitParam,
                              int nPos,
                              QStringList *CNames,
                              int nWrongValue);


    static bool IsUint(QString CValue);


    static bool CheckRecords(QDomNode *CRecords,
                             QList<QStringList> *aCNames,
                             QStringList *aCElementNamesE,
                             QStringList *aCElementNamesR,
                             QStringList aCErrorMessages,
                             QStringList aCTypeVal,
                             int nCurrentUnit);


    static bool CheckRecord(QDomNode *CRecord,
                            QList<QStringList> *aCNames,
                            QStringList *aCElementNamesE,
                            QStringList *aCElementNamesR,
                            QStringList aCErrorMessages,
                            QStringList aCTypeVal,
                            int nCurrentUnit,
                            int nCurrentRecord);


    static void CreateRecordParamsArray(QDomNode* CRecord,

int(&anPlaceArray)[2][6],
                                        QStringList aCTypeVal);


    static bool CheckRecordErrors ( int(&anPlaceArray)[2][6] );


    static void ShowRecordErrors(int(&anPlaceArray)[2][6],
                                 QStringList *aCNames,
                                 int nCurrentUnit,
                                 int nCurrentRecord,
                                 QStringList * aCElementNamesR,
                                 QStringList *aCErrors );


    static bool CheckBitValue(QDomNode *CBit,
                              int(&anPlaceArray)[5],
                              QStringList *aCNames,
                              QStringList *aCElementNamesR,
                              QStringList aCErrorMessages,
                              QStringList aCTypeVal,
                              int nCurrentUnit,
                              int nCurrentRecord,
```

```
                                int nCurrentBit,
                                uint uRecordSize,
                                bool bErrorMessageShow);


public:
    static bool CheckQDocument(QDomDocument *CQDoc,
                               QList<QStringList> *aCNames,
                               QStringList *aCElementNamesE,
                               QStringList *aCElementNamesR,
                               QStringList aCErrorMessages,
                               QStringList aCTypeVal);


    static bool CheckBits(QDomNode *CBits,
                          QList<QStringList> *aCNames,
                          QStringList *aCElementNamesE,
                          QStringList *aCElementNamesR,
                          QStringList aCErrorMessages,
                          QStringList aCTypeVal,
                          int nCurrentUnit,
                          int nCurrentRecord,
                          uint uRecordSize,
                          bool bErrorMessageShow);


    static bool CheckBit(QDomNode *CBit,
                         QList<QStringList> *aCNames,
                         QStringList *aCElementNamesR,
                         QStringList aCErrorMessages,
                         QStringList aCTypeVal,
                         int nCurrentUnit,
                         int nCurrentRecord,
                         int nCurrentBit,
                         uint uRecordSize,
                         bool bErrorMessageShow);


    static bool ValueIsValid(QString CValue, int32_t nP);


    static QDomNode FindChildXml(QString CName, QDomNode
CParentXML);


    static void CheckUnitsCount(QDomDocument *CXDoc,
QList<QStringList> *aCNames, QStringList *acElementNamesE );


    static void CalcMaxSizeRecord(QDomDocument *CXDoc,
QList<QStringList> *aCNames, QStringList *acElementNamesE );
```

```
    static QDomNode CreateModel(QDomDocument *CXDoc,
QList<QStringList> *aCNames, QStringList *acElementNamesE );


    static QDomNode CreateUnit(QDomDocument *CXDoc,
QList<QStringList> *aCNames, QStringList *acElementNamesE );


    static QDomNode CreateRecord(QDomDocument *CXDoc,
QList<QStringList> *aCNames, QStringList *acElementNamesE,
QStringList *aCTypeVal);


    static QDomNode CreateBit(QDomDocument *CXDoc,
QList<QStringList> *aCNames, QStringList *acElementNamesE,
QStringList *aCTypeVal);

};




#endif // XML_CHECK_H
```

Приложение А. 4 – файл «xml_editor.h»

```
#ifndef XML_EDITOR_H
#define XML_EDITOR_H

#include <QMainWindow>
#include <QtXml>
#include <QtWidgets>
#include"tabletree.h"

QT_BEGIN_NAMESPACE
namespace Ui { class XML_Editor; }
QT_END_NAMESPACE

class XML_Editor : public QMainWindow
{
    Q_OBJECT

public:
    XML_Editor(QWidget *parent = nullptr);
    ~XML_Editor();

    // Локальный
    TableTree *tableTree;


private slots:
    void on_action_5_triggered();
```

```
    void on_pushButton_3_clicked();

    void on_treeWidget_currentItemChanged(QTreeWidgetItem
*current, QTreeWidgetItem *previous);

    void on_pushButton_2_clicked();

    void on_checkBox_stateChanged(int arg1);

    void on_pushButton_4_clicked();

    void on_pushButton_clicked();

    void on_pushButton_5_clicked();

    void on_treeWidget_itemClicked(QTreeWidgetItem *item, int
column);

    void ShowContextMenu(const QPoint& pos);

    void ReplaceElementHigher();

    void ReplaceElementLower();

    void AddElementHigher();

    void AddElementLower();

private:
    Ui::XML_Editor *ui;
    int GetNodeLevel(QTreeWidgetItem *CItem);
    QMenu m_CMenu;

    QList<QAction *> m_aCActions =
    {
        new QAction("Добавить потомка"),
        new QAction("Удалить элемент"),
        new QAction("Добавить элемент выше"),
        new QAction("Добавить элемент ниже"),
        new QAction("Переместить элемент выше"),
        new QAction("Переместить элемент ниже"),
    };

    enum Actions {
        A_AddChild,
        A_Remove,
        A_AddHigher,
        A_AddLower,
        A_ReplaceHigher,
        A_ReplaceLower
    };

    // Проверка документа
```

```
    /*-----------------------------------------------------------
-------------------------------------------------------------
--------------------*/

//    bool CheckQDocument(QDomDocument *CQDoc,
QList<QStringList> *aCNames, QStringList *aCElementNamesE);
//    bool CheckUnits(QDomElement *CUnits, QList<QStringList>
*aCNames, QStringList *aCElementNamesE);
//    int CheckUnitsParam(QDomNode *CParam, int nPosition, int
nWrongValueFlag);

//    bool CheckUnit(QDomNode *CUnit, QList<QStringList>
*aCNames, QStringList *aCElementNamesE);
//    int CheckUnitValue(QDomNode *CUnitParam, int nPos,
QStringList *CNames, int nWrongValue);
//    bool ValueIsValid(QString CValue, int32_t nP);

//    bool CheckRecords(QDomNode *CRecords, QList<QStringList>
*aCNames, QStringList *aCElementNamesE);

//    bool CheckRecord(QDomNode *CRecord, QList<QStringList>
*aCNames, QStringList *aCElementNamesE);
//    void CreateRecordParamsArray(QDomNode* CRecord,
int(&anPlaceArray)[2][6]);
//    bool CheckRecordErrors ( int(&anPlaceArray)[2][6] );
//    void ShowRecordErrors(int(&anPlaceArray)[2][6],
QStringList *aCNames, QStringList *aCErrors );

//    bool CheckBits(QDomNode *CBits, QList<QStringList>
*aCNames, QStringList *aCElementNamesE, uint uRecordSize, bool
bErrorMessageShow);

//    bool CheckBit(QDomNode *CBit, QList<QStringList> *aCNames,
QStringList *aCElementNamesE, uint uRecordSize, bool
bErrorMessageShow);
//    bool CheckBitValue(QDomNode *CBit,int(&anPlaceArray)[5],
QStringList *aCNames, uint uRecordSize, bool bErrorMessageShow);

//    bool IsUint(QString CValue);

    /*-----------------------------------------------------------
-------------------------------------------------------------
--------------------*/
    void UpdateTree(QDomDocument* CXDoc, QStringList*
aCElementNamesE, QStringList* aCElementNamesR,
QList<QStringList> *aaCNamesString);
    void UpdateTreeUnitsRecords(QTreeWidgetItem *CCurrentItem,
QDomNode CParentXmlNode, bool bIsUnit, QStringList*
aCElementNamesE, QStringList* aCElementNamesR,
QList<QStringList> *aaCNamesString);
    void UpdateTreeBits(QTreeWidgetItem *CCurrentItem, QDomNode
CXmlBit, QString CXStringName, QString CBitName);
```

```
    QDomNode FindXml(QDomDocument *CXDoc,QTreeWidgetItem
*CSelectedNode, QList<QStringList> *aaCNamesString, QStringList
*aCElementNamesE);
    QDomNode FindUnit(QDomNode *CCurrentXML,int
nCCurrentNodeIndex, QStringList *aCElementNamesE);
    QDomNode FindRecordOrBit(QDomNode *CCurrentXML,int
nCCurrentNodeIndex, QString CContainerName, QString CXmlName);

    QList<QDomNode> *CreateParamList(QList<QDomNode> *aCValues,
QDomNode CParentXmlNode, QString CSNodeName, QList<QStringList>
*aaCNamesString, QStringList *aCElementNamesE);

    void SetupMenu();

    void ClearDocument(QDomDocument* CXDoc, QStringList
*aCElementNamesE,QList<QStringList> *aaCNamesString);
    void ClearUnits(QDomNode CUnits, QStringList
*aCElementNamesE, QList<QStringList> *aaCNamesString);
    void ClearUnit(QDomNode CUnit, QStringList *aCElementNamesE,
QList<QStringList> *aaCNamesString);
    void ClearRecords(QDomNode CRecords, QStringList
*aCElementNamesE, QList<QStringList> *aaCNamesString);
    void ClearRecord(QDomNode CRecord, QStringList
*aCElementNamesE, QList<QStringList> *aaCNamesString);
    void ClearBits(QDomNode CBits, QStringList *aCElementNamesE,
QList<QStringList> *aaCNamesString);
    void ClearBit(QDomNode CBit, QStringList *aCElementNamesE,
QList<QStringList> *aaCNamesString);

public Q_SLOTS:
    void EndOfCheckTable(bool bNeedUpdateTree, bool
bNeedToUpdateMaxSizeRecord);

};
#endif // XML_EDITOR_H
```

Приложение А. 5 – файл «global_vars.cpp»

```
/*_____________________________________________ global_vars.cpp
_____________________________________________ */

/**
    \file       global_vars.cpp
    \brief      Редактор XML
    \author     Прокуров Д.А.
    \version    1.0
    \date       11.02.2020
    \note       Класс содержит глобальные константы и переменные
для функционирования программы

*/
```

```
/*
_____________________________________________________________________________
_____________________________________ */

#include "global_vars.h"


QDomDocument *g_CXDoc = new QDomDocument(); // Текущий документ
XML
QTreeWidgetItem *g_CSelectedTreeItem = new QTreeWidgetItem(); //
Выбранный элемент дерева
QDomNode g_CCurrentParentNode; // Текущий элемент XML
QList<QDomNode> *g_aCXmlParams = new QList<QDomNode>(); //
Список свойств текущего элемента
bool g_bAllowDialogWindow = false; // Признак разрешения
использования диалоговых сообщений

// Константы

const uint g_nMaxByteSize = 32768; // Максимальный размер байта
const uint g_nMaxBitSize = 32768 * 8; // Максимальный размер
бита

// Список названий элементов XML (индексы: g_aElements)
QStringList g_aCElementNamesE = {
  "Units", "Unit", "record", "bit"
};

// Список названий элементов дерева (индексы: g_aElements)
QStringList g_aCElementNamesR = {
  "Модель", "Модуль", "Запись", "Бит"
};

// Список свойств модели (индексы: g_aModelParams)
QStringList g_aCModelParamNames = {
     "CountByteHeader", "CountUnit", "MaxSizeRecord"
};

// Список свойств модуля (индексы: g_aUnitParams)
QStringList g_aCUnitParamNames = {
  "codeOperation", "information", "records"
};

// Список свойств записи (индексы: g_aRecordParams)
QStringList g_aCRecordParamNames = {
    "offset", "size", "string", "type", "bits", "goodvalue"
};

// Список свойств бита (индексы: g_aBitParams)
QStringList g_aCBitParamNames = {
  "offset", "size", "string", "type", "goodvalue"
};
```

```
// Общий список всех свойств (индексы: g_aElements)
QList<QStringList> g_aaCNamesString = {
    g_aCModelParamNames, g_aCUnitParamNames,
g_aCRecordParamNames, g_aCBitParamNames
};

// Названия отображаемых свойств на русском языке (индексы:
g_aRusParams)
QStringList g_aCParamRusNames = {
    "Кол-во байт заголовка", "Кол-во модулей", "Максимальный
размер записи",
    "Код операции", "Смещение", "Размер", "Название", "Тип",
"Хорошее значение"
};

// Список значений свойства "тип" (индексы: g_aValTypes)
QStringList g_aCTypeVal = {
    "d", "x", "b", "mac", "ip"
};

// Список возможных ошибок (TODO: Проверить и дополнить, а также
сделать перечисление для выбора элементов списка) (индексы:
g_aElements)
QStringList g_aCErrorMessages = {
                                                "Ошибка!","Неверный
формат XML-документа","Неверное название корневого эл-та XML-
документа",
                                                "Не удалось загрузить
файл. Файл повреждён или имеет неправильный формат.",
                                                " Проверьте загружаемый
файл.","Неверный формат свойства модели",
                                                "Повторение одного и
того же параметра модели","Обязательное свойство модели
отсутствует",
                                                "Неверное значение
параметра ","Повторение одного и того же параметра
","Отсутствует обязательный элемент ",
                                                "Элементы bits и
goodvalue не могут одновременно существовать в одном блоке.",
                                                "Хорошее значение лежит
в недопустимом диапазоне или имеет неправильный формат.",
                                                "Поле bits не может
присутствовать в блоке не битового типа.",
                                                "В блоке битового типа
обязательно должно быть поле bits.",
                                                "Размер записи типа mac
может равняться только 6.","Размер записи типа ip может
равняться только 4.",
                                                "Сумма размера и
смещения бита не может быть больше размера записи.","Невозможно
добавить потомка биту.",
                                                "Добавить потомка можно
только записи битового типа.","Значение не может быть пустым!",
```

```
                                                    "Хорошее значение в
записи не шестнадцатеричного или десятичного типа недопустимо.",
                                                    "Ошибка сохранения
файла!"
                                                };
```

Приложение А. 6 – файл «main.cpp»

```
#include "xml_editor.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    XML_Editor w;
    w.show();
    return a.exec();
}
```

Приложение А. 7 – файл «tabletree.cpp»

```
#include "tabletree.h"

TableTree::TableTree()
{
    // Список полей свойств модели
    m_aCModelItems = new QList<QTreeWidgetItem *>();

    // Список полей свойств модуля
    m_aCUnitItems = new QList<QTreeWidgetItem *>();

    // Список полей свойств записи
    m_aCRecordItems = new QList<QTreeWidgetItem *>();

    // Список полей свойств бита
    m_aCBitItems = new QList<QTreeWidgetItem *>();

    // Список спин-боксов
    m_aCSpinBoxes = new QList<QSpinBox *>();

    // Список комбо-боксов
    m_aCComboboxes = new QList<QComboBox *>();


    // Устанавливаем количество столбцов
    this->setColumnCount(2);

    // Устанавливаем заголовки
    this->setHeaderLabels({"Название","Значение"});

    // Цикл по количеству свойств модели
    for (int i = 0; i < g_aaCNamesString.at(nEModel).count(); i
++)
```

```
    {
        // Создаём поле таблицы
        QTreeWidgetItem *CNewTreeWidget = new
QTreeWidgetItem(this);

        // Устанавливаем название поля
        CNewTreeWidget->setText(0,g_aCParamRusNames.at(i));

        // Если CountByteHeader
        if(i == nMPCountByteHeader)
        {
            // Создаём спин-бокс, устанавливаем его свойства,
добавляем в поле таблицы и в список и подключаем обработчик
изменения значения
            QSpinBox *CCountByteHeaderSpinBox= new QSpinBox();
            CCountByteHeaderSpinBox-
>setRange(0,(int)g_nMaxByteSize);
            this-
>setItemWidget(CNewTreeWidget,1,CCountByteHeaderSpinBox);
            m_aCSpinBoxes->append(CCountByteHeaderSpinBox);
            connect(CCountByteHeaderSpinBox,static_cast<void
(QSpinBox::*)(const QString
&)>(&QSpinBox::valueChanged),this,&TableTree::SetValueInZeroRow)
;

        }

        // Добавляем поле в список полей и скрываем его
        m_aCModelItems->append(CNewTreeWidget);
        CNewTreeWidget->setHidden(true);
    }

    // Цикл по количеству свойств модуля
    for (int i = 0; i < g_aaCNamesString.at(nEUnit).count() - 1;
i++)
    {
        // Создаём поле таблицы
        QTreeWidgetItem *CNewTreeWidget = new
QTreeWidgetItem(this);

        // Устанавливаем название поля
        if(i == nUPcodeOperation)
            CNewTreeWidget-
>setText(0,g_aCParamRusNames.at(RPcodeOperation));
        else
            CNewTreeWidget-
>setText(0,g_aCParamRusNames.at(RPstring));

        // Добавляем поле в список полей, скрываем его и
разрешаем его изменение
        m_aCUnitItems->append(CNewTreeWidget);
        CNewTreeWidget->setHidden(true);
```

```
        CNewTreeWidget->setFlags(CNewTreeWidget->flags() |
Qt::ItemIsEditable);
    }

    // Цикл по количеству свойств записи
    for (int i = 0; i < g_aaCNamesString.at(nERecord).count() -
1; i++)
    {
        // Создаём поле таблицы
        QTreeWidgetItem *CNewTreeWidget = new
QTreeWidgetItem(this);

        // Устанавливаем название поля
        CNewTreeWidget->setText(0,g_aCParamRusNames.at(i +
RPoffset));

        // Если смещение или размер
        if(i<RPstring - RPoffset)
        {
            // Создаём спин-бокс, устанавливаем его свойства,
добавляем в поле таблицы и подключаем обработчик изменения
значения
            QSpinBox *CRecordSpinBox= new QSpinBox();
            CRecordSpinBox->setRange(0,(int)g_nMaxByteSize);
            this-
>setItemWidget(CNewTreeWidget,1,CRecordSpinBox);
            m_aCSpinBoxes->append(CRecordSpinBox);
            if(i == RPoffset - RPoffset)
                connect(CRecordSpinBox,static_cast<void
(QSpinBox::*)(const QString
&)>(&QSpinBox::valueChanged),this,&TableTree::SetValueInZeroRow)
;
            else
                connect(CRecordSpinBox,static_cast<void
(QSpinBox::*)(const QString
&)>(&QSpinBox::valueChanged),this,&TableTree::SetValueInOneRow);
        }
        // Если тип
        else if(i == RPtype - RPoffset)
        {
            /* Создаём спин-бокс, устанавливаем набор элементов
с установкой текущего, добавляем в поле таблицы
            и подключаем обработчик изменения значения */
            QComboBox *CCombobox = new QComboBox(this);
            for(int j = 0; j < g_aCTypeVal.count(); j++)
                CCombobox->insertItem(j,g_aCTypeVal[j]);
            CCombobox->setCurrentIndex(VType_D);
            this->setItemWidget(CNewTreeWidget,1,CCombobox);
            m_aCComboboxes->append(CCombobox);

connect(CCombobox,&QComboBox::currentTextChanged,this,&TableTree
::SetValueInThreeRow);
        }
```

```
        // Добавляем поле в список полей, скрываем его и
разрешаем его изменение
        CNewTreeWidget->setHidden(true);
        m_aCRecordItems->append(CNewTreeWidget);
        CNewTreeWidget->setFlags(CNewTreeWidget->flags() |
Qt::ItemIsEditable);
    }

    // Цикл по количеству свойств бита
    for (int i = 0; i < g_aaCNamesString.at(nEBit).count(); i++)
    {
        // Создаём поле таблицы
        QTreeWidgetItem *CNewTreeWidget = new
QTreeWidgetItem(this);

        // Устанавливаем название поля
        CNewTreeWidget->setText(0,g_aCParamRusNames.at(i +
RPoffset));

        // Если смещение или размер
        if(i<RPstring - RPoffset)
        {
            // Создаём спин-бокс, устанавливаем его свойства,
добавляем в поле таблицы и подключаем обработчик изменения
значения
            QSpinBox *CBitSpinBox= new QSpinBox();
            CBitSpinBox->setRange(0,(int)g_nMaxBitSize);
            this->setItemWidget(CNewTreeWidget,1,CBitSpinBox);
            m_aCSpinBoxes->append(CBitSpinBox);
            if(i == RPoffset - RPoffset)
                connect(CBitSpinBox,static_cast<void
(QSpinBox::*)(const QString
&)>(&QSpinBox::valueChanged),this,&TableTree::SetValueInZeroRow)
;
            else
                connect(CBitSpinBox,static_cast<void
(QSpinBox::*)(const QString
&)>(&QSpinBox::valueChanged),this,&TableTree::SetValueInOneRow);
        }

        // Если тип
        else if(i == RPtype - RPoffset)
        {
            /* Создаём спин-бокс, устанавливаем набор элементов
с установкой текущего, добавляем в поле таблицы
            и подключаем обработчик изменения значения */
            QComboBox *CCombobox = new QComboBox(this);
            for(int j = 0; j < VType_MAC; j++)
                CCombobox->insertItem(j,g_aCTypeVal[j]);
            CCombobox->setCurrentIndex(VType_D);
            this->setItemWidget(CNewTreeWidget,1,CCombobox);
            m_aCComboboxes->append(CCombobox);
```

```
connect(CCombobox,&QComboBox::currentTextChanged,this,&TableTree
::SetValueInThreeRow);
        }

        // Добавляем поле в список полей, скрываем его и
разрешаем его изменение
        CNewTreeWidget->setHidden(true);
        m_aCBitItems->append(CNewTreeWidget);
        CNewTreeWidget->setFlags(CNewTreeWidget->flags() |
Qt::ItemIsEditable);
    }

    // Подключаем обработчик события изменения значения ячейки

connect(this,&QTreeWidget::itemChanged,this,&TableTree::itemChan
ged);

}

void TableTree::UpdateTable(QList<QDomNode> *aCXmlValues,
QString CSNodeName, QStringList *aCElementNamesE)
{
    // Скрываем все поля таблицы
    this->ClearTable();

    // Параметры, отображающие, чем является текущий элемент
    bool bIsModel  = CSNodeName == aCElementNamesE->at(nEModel);
    bool bIsUnit   = CSNodeName == aCElementNamesE->at(nEUnit);
    bool bIsRecord = CSNodeName == aCElementNamesE-
>at(nERecord);
    bool bIsBit    = CSNodeName == aCElementNamesE->at(nEBit);

    // Уровень вложенности выбранного элемента
    int nNodeLevel = aCElementNamesE->indexOf(CSNodeName);

    // Если выбрана модель, показать соответствующие поля
таблицы и загрузить в них значения
    if ( bIsModel )
    {
        for(int i = 0; i < m_aCModelItems->count(); i++)
            m_aCModelItems->at(i)->setHidden(false);

        LoadTableValues(aCXmlValues,m_aCModelItems, nNodeLevel);
    }

    // Если выбран модуль, показать соответствующие поля таблицы
и загрузить в них значения
    else if ( bIsUnit )
    {
        for(int i = 0; i < m_aCUnitItems->count(); i++)
            m_aCUnitItems->at(i)->setHidden(false);
```

```
        LoadTableValues(aCXmlValues,m_aCUnitItems, nNodeLevel);
    }

    // Если выбрана запись
    else if ( bIsRecord )
    {
        // Показать поля таблицы для записи
        for(int i = 0; i < m_aCRecordItems->count(); i++)
            m_aCRecordItems->at(i)->setHidden(false);

        // Параметры, отображающие тип текущей записи
        bool bIsTypeB   = ( aCXmlValues-
>at(nRPtype).childNodes().at(0).nodeValue() ==
g_aCTypeVal.at(VType_B) );
        bool bIsTypeD   = ( aCXmlValues-
>at(nRPtype).childNodes().at(0).nodeValue() ==
g_aCTypeVal.at(VType_D) );
        bool bIsTypeX   = ( aCXmlValues-
>at(nRPtype).childNodes().at(0).nodeValue() ==
g_aCTypeVal.at(VType_X) );
        bool bIsTypeMac = ( aCXmlValues-
>at(nRPtype).childNodes().at(0).nodeValue() ==
g_aCTypeVal.at(VType_MAC) );
        bool bIsTypeIp  = ( aCXmlValues-
>at(nRPtype).childNodes().at(0).nodeValue() ==
g_aCTypeVal.at(VType_IP) );

        // Размер записи
        uint uSize = ( aCXmlValues-
>at(nRPsize).childNodes().at(0).nodeValue().toUInt() );

        // Если тип и размер являются допустимыми для хорошего
значения, отобразить поле хорошее значение, иначе скрыть
        if ( bIsTypeB || ( ( bIsTypeD || bIsTypeX ) && ( uSize
== 0 || uSize > 4 ) ) || bIsTypeMac || bIsTypeIp )
            m_aCRecordItems->at(nRPgoodvalue - 1)-
>setHidden(true);
        else
            m_aCRecordItems->at(nRPgoodvalue - 1)-
>setHidden(false);

        // Запретить редактирование размера, если тип mac или
ip, в противном случае разрешить
        if ( bIsTypeMac || bIsTypeIp )
            m_aCRecordItems->at(nRPsize)-
>setFlags(m_aCRecordItems->at(nRPsize)->flags() ^
Qt::ItemIsEditable);
        else
            m_aCRecordItems->at(nRPsize)-
>setFlags(m_aCRecordItems->at(nRPsize)->flags() |
Qt::ItemIsEditable);

        // Загрузить значения в поля таблицы
```

```
        LoadTableValues(aCXmlValues,m_aCRecordItems,
nNodeLevel);
    }

    // Если выбран бит
    else if ( bIsBit )
    {
        // Показать поля таблицы для записи
        for(int i = 0; i < m_aCBitItems->count(); i++)
            m_aCBitItems->at(i)->setHidden(false);

        // Размер бита
        uint nSize   = ( aCXmlValues->at(nBPsize
).childNodes().at(0).nodeValue().toUInt() ); // Размер

        // Если размер лежит в диапазоне от 1 до 32, то
добавляем поле для хорошего значения, иначе удаляем
        if ( nSize < 1 || nSize > 32 || nSize == 0 )
            m_aCBitItems->at(nBPgoodvalue)->setHidden(true);
        else
            m_aCBitItems->at(nBPgoodvalue)->setHidden(false);

        // Загрузить значения в поля таблицы
        LoadTableValues(aCXmlValues,m_aCBitItems, nNodeLevel);
    }
}

void TableTree::LoadTableValues(QList<QDomNode> *aCXList,
QList<QTreeWidgetItem *> *aCItems,int nNodeLevel)
{
    // Чистим таблицу
    for(int i = 0; i < aCItems->count(); i++)
        aCItems->at(i)->setText(1,"");

    // Цикл по количеству свойств элемента
    for(int i = 0; i < aCXList->count(); i++)
    {
        // Значение свойства
        QString CValue = aCXList-
>at(i).childNodes().at(0).nodeValue();

        // Если виджет не пустой
        if(this->itemWidget(aCItems->at(i),1) != nullptr)
        {
            // Если элемент это...
            switch (nNodeLevel)
            {
                // Модель
                case nEModel:

                    // Если это CountByteHeader, записать
значение во вложенный виджет
                    if(i == nMPCountByteHeader)
```

```
                        m_aCSpinBoxes->at(0)-
>setValue(CValue.toInt());
                    break;

                // Модуль
                case nEUnit:
                    break;

                // Запись
                case nERecord:

                    // Если это размер или смещение, записать
значение во вложенный виджет
                    if(i < nRPstring)
                        m_aCSpinBoxes->at(i+1)-
>setValue(CValue.toInt());

                    // Если это тип
                    else if (i == nRPtype)
                    {
                        // Вложенный комбо-бокс
                        QComboBox *cb;
                        cb = (QComboBox*)this-
>itemWidget(aCItems->at(i),1);

                        // Цикл по количеству возможных значений
типа
                        for(int i = 0; i < g_aCTypeVal.count();
i++)
                        {
                            // Если значение соответствует i-му
значению типа, то выбрать этот элемент в комбо-боксе
                            if(g_aCTypeVal.at(i) == CValue)
                            {
                                cb->setCurrentIndex(i);
                                break;
                            }
                        }
                    }
                    break;

                // Бит
                case nEBit:

                    // Если это размер или смещение, записать
значение во вложенный виджет
                    if(i < nBPstring)
                        m_aCSpinBoxes->at(3+i)-
>setValue(CValue.toInt());

                    // Если это тип
                    else if (i == nBPtype)
                    {
```

```
                            // Вложенный комбо-бокс
                            QComboBox *cb;
                            cb = (QComboBox*)this-
>itemWidget(aCItems->at(i),1);

                            // Цикл по количеству возможных значений
типа
                            for(int i = 0; i < g_aCTypeVal.count();
i++)
                            {
                                // Если значение соответствует i-му
значению типа, то выбрать этот элемент в комбо-боксе
                                if(g_aCTypeVal.at(i) == CValue)
                                {
                                    cb->setCurrentIndex(i);
                                    break;
                                }
                            }
                        }
                        break;
                }
            }

            // Записать значение в поле таблицы
            aCItems->at(i)->setText(1,CValue);
        }

}



void TableTree::SetValueInZeroRow(const QString& value)
{
    // Индекс искомого поля
    int nFindIndex = 0;

    // Цикл по количеству полей таблицы
    for(int i = 0; i < this->topLevelItemCount();i++)
    {
        // Если поле видимое, сохраняем его индекс и выходим из
цикла
        if(!this->topLevelItem(i)->isHidden())
        {
            nFindIndex = i;
            break;
        }
    }

    // Сохраняем значение в скрытое виджетом поле таблицы
    this->topLevelItem(nFindIndex)->setText(1, value);
}
```

```
void TableTree::SetValueInOneRow(const QString& value)
{
    // Индекс искомого поля
    int nFindIndex = 0;

    // Цикл по количеству полей таблицы
    for(int i = 0; i < this->topLevelItemCount();i++)
    {
        // Если поле видимое, сохраняем индекс следующего и
выходим из цикла
        if(!this->topLevelItem(i)->isHidden())
        {
            nFindIndex = i + 1;
            break;
        }
    }

    // Сохраняем значение в скрытое виджетом поле таблицы
    this->topLevelItem(nFindIndex)->setText(1, value);
}

void TableTree::SetValueInThreeRow(const QString& value)
{
    // Индекс искомого поля
    int nFindIndex = 0;

    // Цикл по количеству полей таблицы
    for(int i = 0; i < this->topLevelItemCount();i++)
    {
        // Если поле видимое, сохраняем индекс идущего за ним
через 3 и выходим из цикла
        if(!this->topLevelItem(i)->isHidden())
        {
            nFindIndex = i + 3;
            break;
        }
    }

    // Сохраняем значение в скрытое виджетом поле таблицы
    this->topLevelItem(nFindIndex)->setText(1, value);
}


void TableTree::itemChanged(QTreeWidgetItem *item, int column)
{
    // Параметр необходимости обновления дерева
    bool bNeedToUpdateTree = false;

    // Параметр необходимости обновления значения максимального
размера записи
    bool bNeedToUpdateMaxSizeRecord = false;

    // Если изменилось значение свойства
```

```
    if(column == 1)
    {
        // Индекс свойства
        int nChangedItemIndex = this->indexOfTopLevelItem(item);

        // Смещение индекса свойства
        int nTotalOffset = 0;

        // Установка смещения индекса
        if(g_aCXmlParams->at(0).parentNode().nodeName() ==
g_aCElementNamesE.at(nEUnit))
            nTotalOffset = ICodeOperation;
        else if(g_aCXmlParams->at(0).parentNode().nodeName() ==
g_aCElementNamesE.at(nERecord))
            nTotalOffset = IROffset;
        else if(g_aCXmlParams->at(0).parentNode().nodeName() ==
g_aCElementNamesE.at(nEBit))
            nTotalOffset = IBOffset;


        // Если изменяемое свойство относится к списку свойств,
в которых не требуются специальные проверки
        if(item->text(0) ==
g_aCParamRusNames.at(RPCountByteHeader) ||
          (item->text(0) == g_aCParamRusNames.at(RPoffset) &&
g_aCXmlParams->at(0).parentNode().nodeName() ==
g_aCElementNamesE.at(nERecord))  ||
           item->text(0) == g_aCParamRusNames.at(RPstring))
        {
            // Если это свойство "Название" или значение
свойства не пустое
            if(item->text(1) != "" || item->text(0) ==
g_aCParamRusNames.at(RPstring))
            {
                // Записываем значение свойства
                g_aCXmlParams->at(nChangedItemIndex -
nTotalOffset).childNodes().at(0).setNodeValue(item->text(1));

                // Если это свойство "Название", то необходимо
изменить дерево
                if (item->text(0) ==
g_aCParamRusNames.at(RPstring))
                    bNeedToUpdateTree = true;
            }

            // Иначе выводим ошибку и восстанавливаем предыдущее
значение
            // TODO: проверить на возможность выпила
дополнительного условия
            else if(item->text(0) !=
g_aCParamRusNames.at(RPstring))
            {
                QErrorMessage CErr;
```

```
                CErr.showMessage("Значение не может быть
пустым!","Ошибка!");
                CErr.exec();
                item->setText(1,g_aCXmlParams->at(this-
>indexOfTopLevelItem(item)).childNodes().at(0).nodeValue());

            }
        }

        else
        {
            // Если код операции
            if(item->text(0) ==
g_aCParamRusNames.at(RPcodeOperation))
            {
                // Если значение пустое или неверный формат кода
операции, выводим ошибку, иначе записываем значение
                if(item->text(1).isEmpty() ||
!XML_Check::ValueIsValid(item->text(1),8))
                {
                    QErrorMessage CErr;
                    CErr.showMessage("Неверное значение
ячейки","Ошибка!");
                    CErr.exec();
                    item->setText(1,g_aCXmlParams-
>at(nChangedItemIndex -
nTotalOffset).childNodes().at(0).nodeValue());
                }

                else g_aCXmlParams->at(nChangedItemIndex -
nTotalOffset).childNodes().at(0).setNodeValue(item->text(1));
            }

            // Если тип
            else if(item->text(0) ==
g_aCParamRusNames.at(RPtype))
            {
                // Если это запись
                if(g_aCXmlParams->at(0).parentNode().nodeName()
== g_aCElementNamesE.at(nERecord))
                {

                    // Если тип был битовый, стал не битовый или
наоборот
                    if((item->text(1) == g_aCTypeVal.at(VType_B)
&&
                       g_aCXmlParams->at(nChangedItemIndex -
nTotalOffset).childNodes().at(0).nodeValue() !=
g_aCTypeVal.at(VType_B)) ||
                       (item->text(1) != g_aCTypeVal.at(VType_B)
&&
```

```
                        g_aCXmlParams->at(nChangedItemIndex -
nTotalOffset).childNodes().at(0).nodeValue() ==
g_aCTypeVal.at(VType_B)))
                    {

                        // Если был не битовый
                        if(g_aCXmlParams->at(nChangedItemIndex -
nTotalOffset).childNodes().at(0).nodeValue() !=
g_aCTypeVal.at(VType_B))
                        {
                            //Если есть goodvalue, удаляем его
                            if(g_aCXmlParams->count()>4)
                                g_aCXmlParams-
>at(4).parentNode().removeChild(g_aCXmlParams->at(4));

                            //Создаём поле bits
                            QDomNode CBits = g_CXDoc-
>createElement(g_aaCNamesString.at(nERecord).at(nRPbits));
                            g_aCXmlParams-
>at(0).parentNode().appendChild(CBits);
                        }

                        // Если был битовый
                        else
                        {

                            // Ставим признак необходимости
обновления дерева
                            bNeedToUpdateTree = true;

                            // Удаляем поле bits
                            // TODO: Проверить на возможность
использования функции FindChildXML
                            QDomNode CBits; //=
XML_Check::FindChildXml(g_aaCNamesString.at(nERecord).at(nRPbits
),g_CCurrentParentNode) ;

                            for(int i = 0; i < g_aCXmlParams-
>at(0).parentNode().childNodes().count(); i++)
                            {
                                if(g_aCXmlParams-
>at(0).parentNode().childNodes().at(i).nodeName() ==
g_aaCNamesString.at(nERecord).at(nRPbits))
                                {
                                    CBits = g_aCXmlParams-
>at(0).parentNode().childNodes().at(i);
                                    break;
                                }
                            }

                            g_aCXmlParams-
>at(0).parentNode().removeChild(CBits);
                        }
```

```
                    }

                    // Если mac или ip
                    if(item->text(1) ==
g_aCTypeVal.at(VType_MAC) || item->text(1) ==
g_aCTypeVal.at(VType_IP))
                    {
                        // Ставим признак необходимости
обновления максимального размера записи
                        bNeedToUpdateMaxSizeRecord = true;

                        // Удалить goodvalue
                        // TODO: Проверить на возможность замены
цифр
                        if(g_aCXmlParams->count()>4)
                            g_aCXmlParams-
>at(4).parentNode().removeChild(g_aCXmlParams->at(4));

                        // Выставить константный размер и
сделать поле размера недоступным для редактирования
                        if(item->text(1) ==
g_aCTypeVal.at(VType_MAC))
                        {
                            // TODO: Проверить на возможность
замены цифр
                            this->blockSignals(true);
                            g_aCXmlParams-
>at(nRPsize).childNodes().at(0).setNodeValue("6");

                            m_aCSpinBoxes->at(2)->setValue(6);
                            m_aCSpinBoxes->at(2)->setRange(6,6);
                            this->blockSignals(false);
                        }
                        else
                        {
                            this->blockSignals(true);
                            g_aCXmlParams-
>at(nRPsize).childNodes().at(0).setNodeValue("4");

                            m_aCSpinBoxes->at(2)->setValue(4);
                            m_aCSpinBoxes->at(2)->setRange(4,4);
                            this->blockSignals(false);
                        }
                    }

                    // Иначе восстановить диапазон возможных
значений
                    else
                        m_aCSpinBoxes->at(2)-
>setRange(0,(int)g_nMaxByteSize);

                }
```

```
                // Записываем значение
                g_aCXmlParams->at(nChangedItemIndex -
nTotalOffset).childNodes().at(0).setNodeValue(item->text(1));
            }

            // Если хорошее значение
            else if(item->text(0) ==
g_aCParamRusNames.at(RPgoodvalue))
            {
                // Если родительский компонент - запись
                if(g_aCXmlParams->at(0).parentNode().nodeName()
== g_aCElementNamesE.at(nERecord))
                {
                    // Если значение не пустое и имеет
правильный формат
                    if(item->text(1) != "" &&
                       XML_Check::ValueIsValid(item->text(1), 8
* (g_aCXmlParams-
>at(nRPsize).childNodes().at(0).nodeValue().toInt())))
                    {
                        // Если есть поле goodvalue, пишем туда,
иначе создаём и пишем туда
                        if(g_aCXmlParams->count()>4)
                            g_aCXmlParams->at(nRPgoodvalue -
1).childNodes().at(0).setNodeValue(item->text(1));

                        // Если goodvalue нет, создаём его и
пишем туда значение
                        else
                        {
                            QDomNode CGoodvalue = g_CXDoc-
>createElement(g_aaCNamesString.at(nERecord).at(nRPgoodvalue));
                            QDomNode CValueNode = g_CXDoc-
>createTextNode(item->text(1));
                            CGoodvalue.appendChild(CValueNode);
                            g_aCXmlParams-
>at(0).parentNode().appendChild(CGoodvalue);
                        }
                    }

                    // Если значение пустое или неверное
                    else
                    {
                        //Если значение пустое
                        if(item->text(1) == "")
                        {
                            //Если есть goodvalue, удаляем его
                            if(g_aCXmlParams->count()>4)
                                g_aCXmlParams-
>at(4).parentNode().removeChild(g_aCXmlParams->at(4));
                        }
```

```
                        // Если неверное значение, вывести
ошибку и восстановить значение
                        else
                        {
                            QErrorMessage CErr;
                            CErr.showMessage("Неверное значение
goodvalue.","Ошибка!");
                            CErr.exec();

                            if(g_aCXmlParams->count()>4)
                                item->setText(1,g_aCXmlParams-
>at(nRPgoodvalue - 1).childNodes().at(0).nodeValue());
                            else
                                item->setText(1,"");
                        }
                    }
                }

                // Если родительский компонент - бит
                else
                {
                    // Значение не пустое и имеет правильный
формат
                    if(item->text(1) != "" &&
                       XML_Check::ValueIsValid(item->text(1),
(g_aCXmlParams-
>at(nRPsize).childNodes().at(0).nodeValue().toInt())))
                    {
                        // Если есть поле goodvalue, пишем туда,
иначе создаём и пишем туда
                        if(g_aCXmlParams->count()>4)
                            g_aCXmlParams-
>at(nBPgoodvalue).childNodes().at(0).setNodeValue(item-
>text(1));

                        // Если goodvalue нет, создаём его и
пишем туда значение
                        else
                        {
                            QDomNode CGoodvalue = g_CXDoc-
>createElement(g_aaCNamesString.at(nERecord).at(nRPgoodvalue));
                            QDomNode CValueNode = g_CXDoc-
>createTextNode(item->text(1));
                            CGoodvalue.appendChild(CValueNode);
                            g_aCXmlParams-
>at(0).parentNode().appendChild(CGoodvalue);
                        }
                    }

                    // Если значение пустое или неверное
                    else
                    {
                        // Если значение пустое
```

```
                        if(item->text(1) == "")
                        {
                            // Если есть goodvalue, удаляем его
                            if(g_aCXmlParams->count()>4)
                                g_aCXmlParams-
>at(4).parentNode().removeChild(g_aCXmlParams->at(4));
                        }

                        // Если неверное значение, вывести
ошибку и восстановить значение
                        else
                        {
                            QErrorMessage CErr;
                            CErr.showMessage("Неверное значение
goodvalue.","Ошибка!");
                            CErr.exec();

                            if(g_aCXmlParams->count()>4)
                                item->setText(1,g_aCXmlParams-
>at(nBPgoodvalue).childNodes().at(0).nodeValue());
                            else
                                item->setText(1,"");
                        }
                    }

                }
            }

            // Если offset или size бита
            else if(g_aCXmlParams->at(0).parentNode().nodeName()
== g_aCElementNamesE.at(nEBit))
            {
                int nOffset = m_aCSpinBoxes->at(3)->value(); //
Смещение
                int nSize   = m_aCSpinBoxes->at(4)->value(); //
Размер бита
                int nSum    = nOffset + nSize; // Сумма
вышеназванных эл-тов

                QDomNode CParentNode = g_aCXmlParams-
>at(0).parentNode().parentNode().parentNode(); // Родительская
запись
                QDomNode CXmlSize =
XML_Check::FindChildXml(g_aaCNamesString.at(nERecord).at(nRPsize
),CParentNode); // Размер родительской записи
                int nRecordSize =
CXmlSize.childNodes().at(0).nodeValue().toInt(); // Размер
родительской записи

                // Если сумма смещения и размера бита меньше или
равна размеру записи
                if (nSum <= nRecordSize * 8)
                {
```

```
                    // Если goodvalue отсутствует или совместимо
с размером бита, записываем значение
                    if(g_aCXmlParams->count() <= 4 ||
                      (g_aCXmlParams->count() > 4 &&
XML_Check::ValueIsValid(g_aCXmlParams-
>at(4).childNodes().at(0).nodeValue(),nSize)))
                    {
                        g_aCXmlParams->at(this-
>indexOfTopLevelItem(item) -
nTotalOffset).childNodes().at(0).setNodeValue(item-
>text(column));
                    }

                    // Если пользователь действует на свой страх
и риск или решил удалить goodvalue, удалить goodvalue и
сохранить значение
                    else
                    {
                        // Параметр необходимости удаления
goodvalue
                        bool bNeedRemove = true;

                        // Если стоит параметр разрешения
диалоговых окон, вывести окно и сохранить результат
                        if(g_bAllowDialogWindow)
                        {
                            QMessageBox CMsgBox;


CMsgBox.setWindowTitle(QString("Удаление goodvalue").toUtf8());
                            CMsgBox.setText(QString("При задании
этого значения поле Хорошее значение сотрётся.").toUtf8());

CMsgBox.setInformativeText(QString("Вы уверены?").toUtf8());

CMsgBox.setIcon(QMessageBox::Question);
                            CMsgBox.setWindowTitle("Удаление
goodvalue");

CMsgBox.setStandardButtons(QMessageBox::Yes | QMessageBox::No);

                            int nRes = CMsgBox.exec();
                            if(nRes == QMessageBox::No)
                                bNeedRemove = false;
                        }

                        // Если нужно удалить goodvalue, удалить
его
                        if(bNeedRemove)
                        {
                            g_aCXmlParams-
>at(0).parentNode().removeChild(g_aCXmlParams->at(4));
```

```
                                g_aCXmlParams->at(this-
>indexOfTopLevelItem(item) -
nTotalOffset).childNodes().at(0).setNodeValue(item-
>text(column));
                            }
                            // Иначе восстановить значение
                            else
                            {
                                this->blockSignals(true);
                                item->setText(1,g_aCXmlParams-
>at(this->indexOfTopLevelItem(item) -
nTotalOffset).childNodes().at(0).nodeValue());
                                this->blockSignals(false);
                            }
                        }
                    }

                    // Если сумма смещения и размера не
соответствует размеру записи, вывести сообщение об ошибке
                    else
                    {
                        QErrorMessage CErr;
                        CErr.showMessage("Сумма смещения и размера
бита не соответствует размеру записи!","Ошибка!");
                        CErr.exec();
                    }

                }

                // Размер записи
                else if (item->text(0) ==
g_aCParamRusNames.at(RPsize))
                {
                    // Признак необходимости обновления
максимального размера записи
                    bNeedToUpdateMaxSizeRecord = true;

                    uint nSize = item->text(column).toUInt(); //
Значение

                    // Если тип записи битовый
                    if(g_aCXmlParams-
>at(nRPtype).childNodes().at(0).nodeValue() ==
g_aCTypeVal.at(VType_B))
                    {
                        // Ищем поле bits
                        QDomNode CBits =
XML_Check::FindChildXml(g_aaCNamesString.at(nERecord).at(nRPbits
),g_aCXmlParams->at(0).parentNode());

                        // Если все биты соответствуют новому
размеру, записываем его
```

```
if(XML_Check::CheckBits(&CBits,&g_aaCNamesString,&g_aCElementNam
esE,&g_aCElementNamesR,g_aCErrorMessages,g_aCTypeVal,0,0,nSize,f
alse))
                        g_aCXmlParams->at(this-
>indexOfTopLevelItem(item) -
nTotalOffset).childNodes().at(0).setNodeValue(item-
>text(column));

                    // Если пользователь действует на свой страх
и риск или решил удалить биты, удалить биты и сохранить значение
                    else
                    {
                        // Параметр необходимости удаления бит
                        bool bNeedToRemove = true;

                        // Если стоит параметр разрешения
диалоговых окон, вывести окно и сохранить результат
                        if(g_bAllowDialogWindow)
                        {
                            QMessageBox CMsgBox;


CMsgBox.setWindowTitle(QString("Удаление бит").toUtf8());
                            CMsgBox.setText(QString("При задании
этого значения будут удалены биты, не соответствующие размеру
записи.").toUtf8());

CMsgBox.setInformativeText(QString("Вы уверены?").toUtf8());

CMsgBox.setIcon(QMessageBox::Question);
                            CMsgBox.setWindowTitle("Удаление
бит");

CMsgBox.setStandardButtons(QMessageBox::Yes | QMessageBox::No);

                            int nRes = CMsgBox.exec();
                            if(nRes == QMessageBox::No)
                                bNeedToRemove = false;
                        }

                        // Если нужно удалить биты, удалить их
                        if(bNeedToRemove)
                        {
                            // Признак необходимости обновления
дерева
                            bNeedToUpdateTree = true;

                            // Записываем значение
                            g_aCXmlParams->at(this-
>indexOfTopLevelItem(item) -
nTotalOffset).childNodes().at(0).setNodeValue(item-
>text(column));
```

```
                                // Цикл по количеству бит
                                for(int i = 0; i <
CBits.childNodes().count(); i++)
                                {
                                    QDomNode CBit =
CBits.childNodes().at(i);

                                    // Если бит не соответствует
размеру записи, удалить его
                                    if( CBit.nodeName() ==
g_aCElementNamesE.at(nEBit) &&
                                        !XML_Check::CheckBit (
&CBit,

&g_aaCNamesString,

&g_aCElementNamesR,

g_aCErrorMessages,

g_aCTypeVal,

0,0,0,

nSize,
                                                                false
) )
                                    {
                                        CBits.removeChild(CBit);
                                        i--;
                                    }
                                }
                            }

                            // Если не нужно удалить биты,
восстановить предыдущее значение
                            else
                            {
                                this->blockSignals(true);
                                item->setText(1,g_aCXmlParams-
>at(this->indexOfTopLevelItem(item) -
nTotalOffset).childNodes().at(0).nodeValue());
                                this->blockSignals(false);
                            }

                        }
                    }

                    // Если тип не битовый
                    else
                    {
```

```
                    // Если goodvalue есть и не соответствует
размеру
                    if ( g_aCXmlParams->count() > 4 &&
                         !XML_Check::ValueIsValid
(g_aCXmlParams->at(nRPgoodvalue -
1).childNodes().at(0).nodeValue(),
                                                    8 *
(int)nSize ) )
                    {

                        // Признак необходимости удаления
goodvalue
                        bool bNeedRemove = true;

                        // Если стоит параметр разрешения
диалоговых окон, вывести окно и сохранить результат
                        if(g_bAllowDialogWindow)
                        {
                            QMessageBox CMsgBox;


CMsgBox.setWindowTitle(QString("Удаление goodvalue").toUtf8());
                            CMsgBox.setText(QString("При задании
этого значения поле Хорошее значение сотрётся.").toUtf8());

CMsgBox.setInformativeText(QString("Вы уверены?").toUtf8());

CMsgBox.setIcon(QMessageBox::Question);
                            CMsgBox.setWindowTitle("Удаление
goodvalue");

CMsgBox.setStandardButtons(QMessageBox::Yes | QMessageBox::No);

                            int nRes = CMsgBox.exec();
                            if(nRes == QMessageBox::No)
                                bNeedRemove = false;
                        }

                        // Если пользователь действует на свой
страх и риск или решил удалить goodvalue, удалить goodvalue и
сохранить значение
                        if(bNeedRemove)
                        {
                            g_aCXmlParams-
>at(0).parentNode().removeChild(g_aCXmlParams->at(nRPgoodvalue -
1));
                            g_aCXmlParams-
>at(nRPsize).childNodes().at(0).setNodeValue(item-
>text(column));
                        }

                        // Если не нужно удалить goodvalue,
восстановить предыдущее значение
```

```
                        else
                        {
                            this->blockSignals(true);
                            item->setText(1,g_aCXmlParams-
>at(this->indexOfTopLevelItem(item) -
nTotalOffset).childNodes().at(0).nodeValue());
                            this->blockSignals(false);
                        }
                    }

                    // Иначе просто сохранить значение
                    else g_aCXmlParams-
>at(nRPsize).childNodes().at(0).setNodeValue(item-
>text(column));
                }

                // TODO: Добавить функцию вычисления
максимального размера
            }
        }

        // Вызываем сигнал окончания проверки
        emit this->checkIsEnd(bNeedToUpdateTree,
bNeedToUpdateMaxSizeRecord);
    }

    // Защита от изменения названий полей (Если изменяемое поле
является названием), восстанавливаем значение параметра
    else
    {
        if(g_aCXmlParams->at(0).parentNode().nodeName() ==
g_aCElementNamesE.at(nEModel))
            item->setText(0,g_aCParamRusNames.at(item->parent()-
>indexOfChild(item)));
        else if(g_aCXmlParams->at(0).parentNode().nodeName() ==
g_aCElementNamesE.at(nEUnit))
        {
            if(this->indexOfTopLevelItem(item) ==
RPcodeOperation)
                item-
>setText(0,g_aCParamRusNames.at(RPcodeOperation));
            else
                item->setText(0,g_aCParamRusNames.at(RPstring));
        }
        else if(g_aCXmlParams->at(0).parentNode().nodeName() ==
g_aCElementNamesE.at(nERecord))
            item->setText(0,g_aCParamRusNames.at(this-
>indexOfTopLevelItem(item) - 1));
        else
            item->setText(0,g_aCParamRusNames.at(this-
>indexOfTopLevelItem(item) - 6));
    }
}
```

```
void TableTree::ClearTable()
{
    // Цикл по количеству полей таблицы
    for(int i = 0; i < this->topLevelItemCount(); i++)
        // Скрыть поле
        this->topLevelItem(i)->setHidden(true);
}
```

Приложение А. 8 – файл «xml_check.cpp»

```
/*_____________________________________________ xml_check.cpp
_____________________________________________ */

/**
    \file       xml_check.cpp
    \brief      Редактор XML
    \author     Прокуров Д.А.
    \version    1.0
    \date       11.02.2020
    \note       Класс содержит методы обработки и программного
взаимодействия с документом XML

*/

/*
_____________________________________________________________________
_____________________________ */

#include "xml_check.h"

/*
_____________________________________________________________________
_____________________________ */


/*
_____________________________________________________________________
_____________________________ */

        /**
            \fn    bool XML_Check::CheckUnits(QDomElement
*CUnits,
                                              QList<QStringList>
*aCNames,
                                              QStringList
*aCElementNamesE,
                                              QStringList
*aCElementNamesR,
                                              QStringList
aCErrorMessages,
                                              QStringList
aCTypeVal)
```

```
            \brief Функция проверки корневого эл-та XML-
документа

            \param CUnits – корневой эл-т XML-документа

            \param aCNames – массив названий свойств XML

            \param aCElementNamesE – массив названий элементов
XML

            \param aCElementNamesR – массив имён элементов на
русском языке

            \param aCErrorMessages – массив сообщений об ошибке

            \param aCTypeVal – массив значений свойства "type"

            \return Признак правильности эл-та

            \note  Функция проверяет список потомков корневого
эл-та и вызывает функцию проверки
                    модуля, если таковые имеются
        */

bool XML_Check::CheckUnits(QDomElement *CUnits,
                           QList<QStringList> *aCNames,
                           QStringList *aCElementNamesE,
                           QStringList *aCElementNamesR,
                           QStringList aCErrorMessages,
                           QStringList aCTypeVal)
{
    // Счётчик модулей
    int nCurrentModule = 0;

    // Признак правильности свойств модели
    bool bIsUnitsOK = true;

    // Список названий свойств модели
    QStringList aCUnitsNames = aCNames->at(nEModel);

    // Количество свойств модели
    int nParamLength = aCUnitsNames.length();

    // Массив значений и положений эл-тов модели
    int anPlaceArray[nMPMaxSizeRecord + 1] =
{nXPEmptyParametr,nXPEmptyParametr,nXPEmptyParametr};

    // Количество потомков корневого элемента
    int nChildCount = CUnits->childNodes().count();

    // Текст сообщения об ошибке
    QString CErrorMessage = "";
```

```
    try
    {
        // Проход по списку потомков корневого элемента
        for(int nChildCounter = 0; nChildCounter < nChildCount;
nChildCounter++)
        {
            // Текущий потомок
            QDomNode CCurrentNode = CUnits-
>childNodes().item(nChildCounter);

            // Ссылка на текущего потомка
            QDomNode *CPtrCurrentNode = &CCurrentNode;

            // Проход по списку названий
            for(int nNameCounter = 0; nNameCounter <
nParamLength; nNameCounter++)
            {

                bool bIsParamName = (CCurrentNode.nodeName() ==
aCUnitsNames[nNameCounter]);
                bool bIsUnit = (CUnits-
>childNodes().item(nChildCounter).nodeName() == aCElementNamesE-
>at(nEUnit));

                // Соответствует названию
                if( bIsParamName )
                {
                    bool bIsParamNotFind =
(anPlaceArray[nNameCounter] == nXPEmptyParametr);

                    // Если эл-т не найден ранее, проверяем его
на правильность
                    if ( bIsParamNotFind )
                    {
                        int nParamStatus =
CheckUnitsParam(CPtrCurrentNode,nChildCounter,nXPWrongValuePara
etr);
                        anPlaceArray[nNameCounter] =
nParamStatus;

                        bool bIsWrongValue =
(anPlaceArray[nNameCounter] == nXPWrongValueParametr);

                        // Если не признак ошибки, продолжаем
выполнение, иначе вызываем исключение
                        if(!bIsWrongValue) continue;
                        CErrorMessage =
aCErrorMessages.at(EMModelPropFormatError) + " (" + aCNames-
>at(nEModel).at(nNameCounter) + ").";
                        throw new std::exception;

                    } // if ( bIsParamNotFound )
```

```
                    // Если эл-т был ранее, ставим признак
повтора, изменяем сообщение об ошибке и вызываем исключение
                    else
                    {
                        anPlaceArray[nNameCounter] =
nXPRepeatParametr;
                        CErrorMessage =
aCErrorMessages.at(EMModelPropRepeatError) + " (" + aCNames-
>at(nEModel).at(nNameCounter) + ").";
                        throw new std::exception;
                    } // else

                } // ( bIsParamName )

                // Если называется "Unit", то увеличиваем
счётчик модулей и проверяем его правильность
                else if ( bIsUnit )
                {
                    nCurrentModule++;

                    bool bIsUnitOk = CheckUnit (
CPtrCurrentNode,
                                                aCNames,

aCElementNamesE,

aCElementNamesR,

aCErrorMessages,
                                                aCTypeVal,
                                                nCurrentModule
                                                );

                    // Если Unit неправильный, ставим признак
ошибки и выходим из цикла
                    if ( !bIsUnitOk )
                        bIsUnitsOK = false;
                    break;

                } // else if ( bIsUnit )

            } // for ( nNameCounter )

        } // for ( nChildCounter )

        // Проверяем Units на отсутствие обязательных полей
        for(int nNameCounter = 0; nNameCounter < nParamLength;
nNameCounter++)
        {
            bool bIsParamNull = anPlaceArray[nNameCounter] ==
nXPEmptyParametr;
```

```
            // Если обязательное поле есть, продолжаем
выполнение цикла, иначе изменяем сообщение об ошибке и вызываем
исключение
            if( !bIsParamNull ) continue;

            CErrorMessage =
aCErrorMessages.at(EMModelPropNullError) + " (" + aCNames-
>at(nEModel).at(nNameCounter) + ").";
            throw new std::exception;

        } // for ( nNameCounter )

        // Записываем количество модулей
        CUnits-
>childNodes().item(anPlaceArray[nMPCountUnit]).setNodeValue(QStr
ing::number(nCurrentModule));

    } // try

    catch(...)
    {
        // Ставим флаг ошибки и выводим сообщение об ошибке
        bIsUnitsOK = false;
        QErrorMessage CErrMsg;
        CErrMsg.showMessage(CErrorMessage +
aCErrorMessages.at(EMCheckFile),aCErrorMessages.at(EMError));
        CErrMsg.exec();
    }

    return bIsUnitsOK;
}

/*
_____________________________________________________________________
_____________________________ */

        /**
            \fn    int XML_Check::CheckUnitsParam(QDomNode
*CParam, int nPosition, int nWrongValueFlag)

            \param CParam – Проверяемый эл-т

            \param nPosition – Расположение проверяемого эл-та в
списке

            \param nWrongValueFlag – Признак ошибки

            \return Положение эл-та в списке, если его формат
верен, или признак неправильности эл-та

            \note  Функция проверяет формат потомка корневого
эл-та
        */
```

```
int XML_Check::CheckUnitsParam(QDomNode *CParam, int nPosition,
int nWrongValueFlag)
{
    // Возвращаемое значение - признак неверного формата
    int nErr = nWrongValueFlag;

    // Если не содержит лишние поля и значение неотрицательное,
возвращаем положение, иначе признак ошибки
    bool bNodesCount_1  = CParam->childNodes().count() == 1;
    bool bIsNodeType_Text = (bNodesCount_1 && CParam-
>childNodes().item(0).nodeType() == QDomNode::TextNode);
    bool bIsUint = bIsNodeType_Text && IsUint(CParam-
>childNodes().item(0).nodeValue());

    if(bIsUint)
        nErr = nPosition;
    return  nErr;
}

/*
_____________________________________________________________________
_____________________________ */

        /**
            \fn    bool XML_Check::CheckUnit(QDomNode *CUnit,
                                             QList<QStringList>
*aCNames,
                                             QStringList
*aCElementNamesE,
                                             QStringList
*aCElementNamesR,
                                             QStringList
aCErrorMessages,
                                             QStringList aCTypeVal,
                                             int nCurrentModule)

            \brief Функция проверки эл-та XML-документа типа
Unit

            \param CUnit – проверяемый эл-т типа Unit

            \param aCNames – массив названий свойств элементов
телеметрии

            \param aCElementNamesE – массив названий элементов
XML

            \param aCElementNamesR – массив имён элементов на
русском языке

            \param aCErrorMessages – массив сообщений об ошибке
```

```
            \param aCTypeVal - массив значений элемента "type"

            \param nCurrentModule - номер модуля по списку

            \return Признак правильности эл-та

            \note  Функция проверяет список потомков эл-та типа
Unit и вызывает функцию проверки
                    эл-та типа records
        */

bool XML_Check::CheckUnit(QDomNode *CUnit,
                          QList<QStringList> *aCNames,
                          QStringList *aCElementNamesE,
                          QStringList *aCElementNamesR,
                          QStringList aCErrorMessages,
                          QStringList aCTypeVal,
                          int nCurrentModule)
{
    // Признак правильности свойств модуля
    bool bIsUnitOk = true;

    // Массив значений и положений эл-тов модуля
    int anPlaceArray[ nUPrecords + 1 ] = { nXPEmptyParametr,
nXPEmptyParametr, nXPEmptyParametr };

    // Список названий свойств модуля
    QStringList aCUnitsNames = aCNames->at(nEUnit);

    // Количество потомков элемента Unit
    int nNodesCount = CUnit->childNodes().count();

    // Текст сообщения об ошибке
    QString CErrorMessage = "";

    try
    {
        // Проход по списку эл-тов Unit
        for ( int nChildCounter = 0; nChildCounter <
nNodesCount; nChildCounter++ )
        {
            // Текущий параметр
            QDomNode CUnitParam = CUnit-
>childNodes().at(nChildCounter);

            // Ссылка на текущий параметр
            QDomNode *CPtrUnitParam = &CUnitParam;

            // Проход по массиву допустимых названий
            for ( int nNameCounter = 0; nNameCounter <
aCUnitsNames.length(); nNameCounter++ )
            {
```

```
                bool bIsParamNameOk = ( CUnit-
>childNodes().item(nChildCounter).nodeName() == aCUnitsNames[
nNameCounter ] );

                // Если не является параметром
                if( !bIsParamNameOk ) continue;

                bool bIsParamNull = ( anPlaceArray [
nNameCounter ] == nXPEmptyParametr );

                // Если параметр не найден ранее, проверяем его
на правильность
                if ( bIsParamNull )
                {
                    int nParamValue =
CheckUnitValue(CPtrUnitParam,nChildCounter,&aCUnitsNames,nXPWron
gValueParametr);

                    anPlaceArray[nNameCounter] = nParamValue;

                    // Если не признак ошибки, продолжаем
выполнение цикла, иначе вызываем исключение
                    bool bIsParamBad = ( anPlaceArray [
nNameCounter ] == nXPWrongValueParametr );

                    if( !bIsParamBad ) continue;

                    CErrorMessage =
aCErrorMessages.at(EMModulePropFormatError) + " (" + aCNames-
>at(nEUnit).at(nNameCounter) + ").";
                    throw new std::exception;

                } // if ( bIsParamNull )

                else
                {
                    // Если эл-т был ранее, ставим признак
повтора и вызываем исключение
                    anPlaceArray [ nNameCounter ] =
nXPRepeatParametr;
                    CErrorMessage =
aCErrorMessages.at(EMCModulePropRepeatError) + " (" + aCNames-
>at(nEUnit).at(nNameCounter) + ").";
                    throw new std::exception;

                }  // else

            } // for ( nNameCounter )

        } // for ( nChildCounter )


        // Проверяем Unit на отсутствие обязательных полей
```

```
        for ( int nNameCounter = 0; nNameCounter <
aCUnitsNames.length(); nNameCounter++ )
        {
            bool bIsParamNull = ( anPlaceArray [ nNameCounter ]
== nXPEmptyParametr );

            // Если обязательное поле есть, то продолжаем
проверку, иначе вызываем исключение
            if ( !bIsParamNull ) continue;
            CErrorMessage =
aCErrorMessages.at(EMCModulePropNullError) + " (" + aCNames-
>at(nEUnit).at(nNameCounter) + ").";
            throw new std::exception;
        }

        // Если исключений не вызвано, то можно проверять поле
records
        QDomNode CRecords = CUnit->childNodes().item(
anPlaceArray[ nUPrecords ] );
        bIsUnitOk = CheckRecords(&CRecords,
                                 aCNames,
                                 aCElementNamesE,
                                 aCElementNamesR,
                                 aCErrorMessages,
                                 aCTypeVal,
                                 nCurrentModule);
    } // try

    catch(...)
    {
        // Ставим флаг ошибки и выводим сообщение об ошибке
        bIsUnitOk = false;

        QErrorMessage CErrMsg;
        CErrMsg.showMessage(CErrorMessage +
                            " (" + QString::number(
nCurrentModule ) + " " + aCElementNamesR->at( nEUnit ) + " )." +
                           aCErrorMessages.at( EMCheckFile
),aCErrorMessages.at( EMError ));
        CErrMsg.exec();
    }

    return bIsUnitOk;
}

/*
_____________________________________________________________
___________ */

/**
    \fn      int XML_Check::CheckUnitValue(QDomNode *CUnitParam,
int nPos, QStringList *CNames, int nWrongValue)
```

```
    \brief   Функция проверки формата параметров эл-та XML-
документа типа Unit

    \param   CUnitParam – проверяемый параметр эл-та типа Unit

    \param   nPos – Расположение проверяемого эл-та в списке

    \param   CNames – массив названий свойств элементов
телеметрии

    \param   nWrongValue – Признак ошибки

    \return Положение эл-та в списке, если его формат верен, или
признак неправильности эл-та

    \note    Функция проверяет формат потомка эл-та типа Unit
*/

int XML_Check::CheckUnitValue(QDomNode *CUnitParam, int nPos,
QStringList *CNames, int nWrongValue)
{
    bool bIsCodeOperation = CUnitParam->nodeName() == CNames-
>at(nUPcodeOperation);
    bool bIsInformation = CUnitParam->nodeName() == CNames-
>at(nUPinformation);
    bool bHasNotNullValue = CUnitParam->childNodes().count() !=
1;
    bool bHasValue = CUnitParam->childNodes().count() > 0;
    bool bCodeOperationValueBad = ( bHasNotNullValue || !(
CUnitParam->childNodes().item(0).nodeType() ==
QDomNode::TextNode ) );
    bool bInformationValueBad   = ( bHasValue && !( CUnitParam-
>childNodes().item(0).nodeType() == QDomNode::TextNode ) );

    // Если эл-т codeOperation имеет внутренние XML или
значение, не соответствующее правилам (см. след. функцию) или
    // эл-т information имеет внутренние XML, возвращаем признак
ошибки, иначе положение
    if ( ( bIsCodeOperation && ( bCodeOperationValueBad ||
!ValueIsValid ( CUnitParam->childNodes().at(0).nodeValue(), 8 )
) ) ||
          ( bIsInformation && bInformationValueBad ) )
        return nWrongValue;
    else return nPos;
}

/*
_____________________________________________________________
___________ */

/**
    \fn      bool XML_Check::IsUint(QString CValue)
```

```
    \brief   Функция проверки строки на соответствие типу uint

    \param   CValue - проверяемая строка

    \return True если строка соответствует uint, False если не
соответствует

    \note    Функция проверяет строку на соответствие типу uint
*/

bool XML_Check::IsUint(QString CValue)
{
    // Признак соответствия строки типу uint
    bool bConversionResult = true;

    // Преобразование строки в uint
    CValue.toUInt(&bConversionResult,10);

    return bConversionResult;
}

bool XML_Check::CheckRecords(QDomNode *CRecords,
                             QList<QStringList> *aCNames,
                             QStringList *aCElementNamesE,
                             QStringList* aCElementNamesR,
                             QStringList aCErrorMessages,
                             QStringList aCTypeVal,
                             int nCurrentUnit)
{
    // Счётчик записей
    int nCurrentRecord = 0;
    // Признак правильности свойств records
    bool bRecordsOK = true;

    for(int nChildCounter = 0; nChildCounter < CRecords-
>childNodes().count(); nChildCounter++ )
    {
        QDomNode CRecord = CRecords-
>childNodes().item(nChildCounter);
        QDomNode *CPtrRecord = &CRecord;
        bool bIsNameRecord = CRecord.nodeName() ==
aCElementNamesE->at(nERecord);
        // Если поле называется "record"
        if ( bIsNameRecord )
        {
            // Увеличиваем счётчик записей
            nCurrentRecord++;
            // Проверить запись
            bRecordsOK =
CheckRecord(CPtrRecord,aCNames,aCElementNamesE,aCElementNamesR,a
CErrorMessages,aCTypeVal,nCurrentUnit,nCurrentRecord);

            if(!bRecordsOK) break;
```

```
        }
    }
    return bRecordsOK;
}

bool XML_Check::CheckRecord(QDomNode *CRecord,
                            QList<QStringList> *aCNames,
                            QStringList *aCElementNamesE,
                            QStringList *aCElementNamesR,
                            QStringList aCErrorMessages,
                            QStringList aCTypeVal,
                            int nCurrentUnit,
                            int nCurrentRecord)
{
    // Признак правильности свойств record
    bool bRecordOK = true;
    // Массив значений и положений эл-тов записи
    int anPlaceArray[2][6] =
    {
        {

nXPEmptyParametr,nXPEmptyParametr,nXPEmptyParametr,nXPEmptyParam
etr,nXPEmptyParametr,nXPEmptyParametr
        },
        {

nXPEmptyParametr,nXPEmptyParametr,nXPEmptyParametr,nXPEmptyParam
etr,nXPEmptyParametr,nXPEmptyParametr
        }
    };

    QStringList aCRecordNames = aCNames->at(nERecord);

    int nRecordChildCounter = CRecord->childNodes().count();
    // Проход по списку эл-тов record
    for ( int nChildCounter = 0; nChildCounter <
nRecordChildCounter; nChildCounter++ )
    {
        // Проход по массиву допустимых названий
        for ( int nNameCounter = 0; nNameCounter <
aCRecordNames.length(); nNameCounter++ ) //
ПРОВЕРИТЬ!!!!!!!!!!!!!!!
        {
            bool bIsParamNameOk = ( CRecord-
>childNodes().item(nChildCounter).nodeName() == aCRecordNames [
nNameCounter ] );
            bool bIsParamNull   = ( anPlaceArray[ 0 ][
nNameCounter ]    == nXPEmptyParametr );
            // Если не соответствует названию, пропускаем
элемент
            if ( !bIsParamNameOk ) continue;
```

```
            // Если элемента не было, записать положение, иначе
признак повтора
            if ( bIsParamNull )
                anPlaceArray [ 0 ][ nNameCounter ] =
nChildCounter;
            else
                anPlaceArray [ 0 ][ nNameCounter ] =
nXPRepeatParametr;
        }
    }
    // Функция формирования массива признаков
    CreateRecordParamsArray(CRecord,anPlaceArray,aCTypeVal);

    // Функция проверки массива признаков
    bRecordOK = CheckRecordErrors(anPlaceArray);

    if(bRecordOK)
    {
        bool bHasBits = ( anPlaceArray [ 1 ][ nRPbits ] ==
nROVRight );
        if(bHasBits)
        {
            uint uRecordSize = CRecord-
>childNodes().at(anPlaceArray [ 0 ][ nRPsize
]).childNodes().at(0).nodeValue().toUInt();
            QDomNode CRecordSize = CRecord-
>childNodes().at(anPlaceArray [ 0 ][ nRPbits ]);

            // Функция проверки bits
            bRecordOK = CheckBits( &CRecordSize, aCNames,
aCElementNamesE,aCElementNamesR,aCErrorMessages,aCTypeVal,nCurre
ntUnit,nCurrentRecord, uRecordSize, true);
        }
    }
    else
    {
        // Функция вывода ошибок

ShowRecordErrors(anPlaceArray,&aCRecordNames,nCurrentUnit,nCurre
ntRecord,aCElementNamesR,&aCErrorMessages);
    }
    return bRecordOK;
}

void XML_Check::CreateRecordParamsArray(QDomNode* CRecord,
int(&anPlaceArray)[2][6], QStringList aCTypeVal)
{
    // Цикл обработки массива
    for ( int nChildCounter = 0; nChildCounter < 6;
nChildCounter++ )
    {
        bool bBits = nChildCounter == nRPbits;
        bool bString = nChildCounter == nRPstring;
```

```
        bool bBitsFind = anPlaceArray [ 0 ][ nRPbits ] !=
nXPEmptyParametr;

        bool bIsStringFind = anPlaceArray [ 0 ][ nRPstring ] >
nXPEmptyParametr;

        bool bIsStringEmpty = bIsStringFind && CRecord-
>childNodes().item(anPlaceArray [ 0 ][ nRPstring
]).childNodes().count() == 0;

        bool bIsStringNotEmpty = bIsStringFind &&
                                                ( CRecord-
>childNodes().item(anPlaceArray [ 0 ][ nRPstring
]).childNodes().count() == 1 &&
                                                  CRecord-
>childNodes().item(anPlaceArray [ 0 ][ nRPstring
]).childNodes().item(0).nodeType() ==

QDomNode::TextNode );

        // Не bits и не string
        if ( !bBits && !bString )
        {
            bool bIsParamValueCorrect = ( anPlaceArray [ 0 ][
nChildCounter ] >= 0 &&
                    ( CRecord->childNodes().item(anPlaceArray [
0 ][ nChildCounter ]).childNodes().count() == 1 &&
                      CRecord->childNodes().item( anPlaceArray [
0 ][ nChildCounter ]).childNodes().item( 0 ).nodeType() ==
QDomNode::TextNode ) );

            bool bIsGoodvalueNotFind = ( nChildCounter ==
nRPgoodvalue &&

anPlaceArray [ 0 ][ nRPgoodvalue ] == nXPEmptyParametr );

            // Элемент не имеет потомков и внутренних XML
            if ( bIsParamValueCorrect )
            {
                // Значение поля
                QString CValue = CRecord->childNodes().at(
anPlaceArray [ 0 ][
nChildCounter]).childNodes().at(0).nodeValue();
                // Результат парсинга значения текущего эл-та
XML
                uint uOutput = 0;
                bool bIsUint;
                uOutput = CValue.toUInt(&bIsUint,10);

                bool bIsNotGoodvalue = ( nChildCounter !=
nRPgoodvalue );
                // Обязательные поля
```

```
                if ( bIsNotGoodvalue )
                {
                    bool bIsOffset = nChildCounter == nRPoffset;
                    bool bIsSize   = nChildCounter == nRPsize;
                    bool bIsType   = nChildCounter == nRPtype;

                    // Эл-т offset имеет правильный формат
                    if ( bIsOffset && bIsUint && (uOutput <=
g_nMaxByteSize) )
                        anPlaceArray [ 1 ][ nChildCounter ] =
nROVRight;

                    // Эл-т size имеет правильный формат
                    if ( bIsSize && bIsUint )
                    {
                        bool bSizeConcrete = (( uOutput >=
nRSVSize_1byte   &&
                                               uOutput <=
nRSVSize_4byte ) ||
                                               uOutput ==
nRSVSize_6byte );

                        // Запись значений size
                        if ( bSizeConcrete )
                            anPlaceArray [ 1 ][ nChildCounter] =
(int)uOutput;
                        else
                        {
                            if(uOutput <= g_nMaxByteSize)
                                anPlaceArray [ 1 ][
nChildCounter ] = nRSVSize_0_5byte;
                        }
                    }

                    // Тип
                    if(bIsType)
                    {
                        bool bIsType_x   = CValue == aCTypeVal [
VType_X ];
                        bool bIsType_d   = CValue == aCTypeVal [
VType_D ];
                        bool bIsType_b   = CValue == aCTypeVal [
VType_B ];
                        bool bIsType_mac = CValue == aCTypeVal [
VType_MAC ];
                        bool bIsType_ip  = CValue == aCTypeVal [
VType_IP ];

                        // Десятичный или шестнадцатеричный
                        if ( bIsType_x || bIsType_d )
                            anPlaceArray [ 1 ][ nChildCounter ]
= nRTVType_d_x;
                        // Битовый
```

```
                        else if ( bIsType_b )
                            anPlaceArray [ 1 ][ nChildCounter ]
= nRTVType_b;
                        // mac
                        else if ( bIsType_mac )
                            anPlaceArray [ 1 ][ nChildCounter ]
= nRTVType_Mac;
                        // ip
                        else if ( bIsType_ip )
                            anPlaceArray [ 1 ][ nChildCounter ]
= nRTVType_IP;
                    }
                }
                // Поле Goodvalue
                else
                {
                    bool bIsSizeUint;
                    uint nRecordSize = 0;
                    QString CSizeValue = CRecord-
>childNodes().item(anPlaceArray[0][nRPsize]).childNodes().at(0).
nodeValue();
                    nRecordSize =
CSizeValue.toUInt(&bIsSizeUint,10);

                    bool bIsGoodvalueValid     = ValueIsValid (
CValue, (int)nRecordSize * 8 );
                    bool bIsSizeNotWrongOrNull = anPlaceArray [
1 ][ nRPsize ] != nXPEmptyParametr;
                    // Проверка значения
                    if ( bIsSizeNotWrongOrNull &&
bIsGoodvalueValid )
                        // Запись значения
                        anPlaceArray [ 1 ][ nChildCounter ] =
(int)nRecordSize;
                }

            }
            else if(bIsGoodvalueNotFind)
                anPlaceArray [ 1 ][ nRPgoodvalue ] =
nRGVRecordGoodvalueNotFind;
        }
        // Записать признак расположения bits
        else if ( bBits && bBitsFind )
            anPlaceArray [ 1 ][ nRPbits ] = nROVRight;

        // Имя
        else if ( bString && bIsStringFind && ( bIsStringEmpty
|| bIsStringNotEmpty ) )
            anPlaceArray [ 1 ][ nChildCounter ] = nROVRight;

    }
}
```

```
bool XML_Check::CheckRecordErrors ( int(&anPlaceArray)[2][6] )
{
    // Флаг правильности
    bool bRecordOK = false;
    // Массив правильных значений
    int anRightValues[24][6] =
    {
        // Наборы не битового типа:
                                                    {
                                                        // Размер 0,
5 или больше 6 байт
                                                        nROVRight,

nRSVSize_0_5byte,
                                                        nROVRight,
                                                        nRTVType_d_x,
                                                        nROVBitsNull,

nRGVRecordGoodvalueNotFind
                                                    },
                                                    {
                                                        // Размер 1
байт, нет хорошего значения
                                                        nROVRight,

nRSVSize_1byte,
                                                        nROVRight,
                                                        nRTVType_d_x,
                                                        nROVBitsNull,

nRGVRecordGoodvalueNotFind
                                                    },
                                                    {
                                                        // Размер 1
байт, есть хорошее значение
                                                        nROVRight,

nRSVSize_1byte,
                                                        nROVRight,
                                                        nRTVType_d_x,
                                                        nROVBitsNull,

nRGVRecordGoodvalue_8bit
                                                    },
                                                    {
                                                        // Размер 2
байта, нет хорошего значения
                                                        nROVRight,

nRSVSize_2byte,
                                                        nROVRight,
                                                        nRTVType_d_x,
                                                        nROVBitsNull,
```

```
nRGVRecordGoodvalueNotFind
                                                    },
                                                    {
                                                        // Размер 2
байта, хорошее значение меньше 8 бит
                                                        nROVRight,

nRSVSize_2byte,
                                                        nROVRight,
                                                        nRTVType_d_x,
                                                        nROVBitsNull,

nRGVRecordGoodvalue_8bit
                                                    },
                                                    {
                                                        // Размер 2
байта, хорошее значение меньше 16, но больше 8 бит
                                                        nROVRight,

nRSVSize_2byte,
                                                        nROVRight,
                                                        nRTVType_d_x,
                                                        nROVBitsNull,

nRGVRecordGoodvalue_16bit
                                                    },
                                                    {
                                                        // Размер 3
байта, нет хорошего значения
                                                        nROVRight,

nRSVSize_3byte,
                                                        nROVRight,
                                                        nRTVType_d_x,
                                                        nROVBitsNull,

nRGVRecordGoodvalueNotFind
                                                    },
                                                    {
                                                        // Размер 3
байта, хорошее значение меньше 8 бит
                                                        nROVRight,

nRSVSize_3byte,
                                                        nROVRight,
                                                        nRTVType_d_x,
                                                        nROVBitsNull,

nRGVRecordGoodvalue_8bit
                                                    },
                                                    {
```

```
                                                        // Размер 3
байта, хорошее значение меньше 16, но больше 8 бит
                                                        nROVRight,

nRSVSize_3byte,
                                                        nROVRight,
                                                        nRTVType_d_x,
                                                        nROVBitsNull,

nRGVRecordGoodvalue_16bit
                                                    },
                                                    {
                                                        // Размер 3
байта, хорошее значение меньше 24, но больше 16 бит
                                                        nROVRight,

nRSVSize_3byte,
                                                        nROVRight,
                                                        nRTVType_d_x,
                                                        nROVBitsNull,

nRGVRecordGoodvalue_24bit
                                                    },
                                                    {
                                                        // Размер 4
байта, нет хорошего значения
                                                        nROVRight,

nRSVSize_4byte,
                                                        nROVRight,
                                                        nRTVType_d_x,
                                                        nROVBitsNull,

nRGVRecordGoodvalueNotFind
                                                    },
                                                    {
                                                        // Размер 4
байта, хорошее значение меньше 8 бит
                                                        nROVRight,

nRSVSize_4byte,
                                                        nROVRight,
                                                        nRTVType_d_x,
                                                        nROVBitsNull,

nRGVRecordGoodvalue_8bit
                                                    },
                                                    {
                                                        // Размер 4
байта, хорошее значение меньше 16, но больше 8 бит
                                                        nROVRight,

nRSVSize_4byte,
```

```
                                                    nROVRight,
                                                    nRTVType_d_x,
                                                    nROVBitsNull,

nRGVRecordGoodvalue_16bit
                                                },
                                                {
                                                    // Размер 4
байта, хорошее значение меньше 24, но больше 16 бит
                                                    nROVRight,

nRSVSize_4byte,
                                                    nROVRight,
                                                    nRTVType_d_x,
                                                    nROVBitsNull,

nRGVRecordGoodvalue_24bit
                                                },
                                                {
                                                    // Размер 4
байта, хорошее значение меньше 32, но больше 24 бит
                                                    nROVRight,

nRSVSize_4byte,
                                                    nROVRight,
                                                    nRTVType_d_x,
                                                    nROVBitsNull,

nRGVRecordGoodvalue_32bit
                                                },
                                                {
                                                    // Размер 6
байт, нет хорошего значения
                                                    nROVRight,

nRSVSize_6byte,
                                                    nROVRight,
                                                    nRTVType_d_x,
                                                    nROVBitsNull,

nRGVRecordGoodvalueNotFind
                                                },
                                                // Наборы
битового типа (для всех размеров нет хорошего значения, но есть
поле bits)
                                                {
                                                    nROVRight,

nRSVSize_0_5byte,
                                                    nROVRight,
                                                    nRTVType_b,
                                                    nROVRight,
```

```
    nRGVRecordGoodvalueNotFind
                                                    },
                                                    {
                                                        nROVRight,

    nRSVSize_1byte,
                                                        nROVRight,
                                                        nRTVType_b,
                                                        nROVRight,

    nRGVRecordGoodvalueNotFind
                                                    },
                                                    {
                                                        nROVRight,

    nRSVSize_2byte,
                                                        nROVRight,
                                                        nRTVType_b,
                                                        nROVRight,

    nRGVRecordGoodvalueNotFind
                                                    },
                                                    {
                                                        nROVRight,

    nRSVSize_3byte,
                                                        nROVRight,
                                                        nRTVType_b,
                                                        nROVRight,

    nRGVRecordGoodvalueNotFind
                                                    },
                                                    {
                                                        nROVRight,

    nRSVSize_4byte,
                                                        nROVRight,
                                                        nRTVType_b,
                                                        nROVRight,

    nRGVRecordGoodvalueNotFind
                                                    },
                                                    {
                                                        nROVRight,

    nRSVSize_6byte,
                                                        nROVRight,
                                                        nRTVType_b,
                                                        nROVRight,

    nRGVRecordGoodvalueNotFind
                                                    },
```

```
                                                    // Наборы типа ip
и mac (4 для ip, 6 для mac, нет бит и хорошего значения)
                                                    {
                                                        nROVRight,

nRSVSize_4byte,
                                                        nROVRight,
                                                        nRTVType_IP,
                                                        nROVBitsNull,

nRGVRecordGoodvalueNotFind
                                                    },

                                                    {
                                                        nROVRight,

nRSVSize_6byte,
                                                        nROVRight,
                                                        nRTVType_Mac,
                                                        nROVBitsNull,

nRGVRecordGoodvalueNotFind
                                                    }
    };

    // Если массив значений записи соответствует одному эл-ту из
набора правильных значений, то возвращаем true
    for ( int nArrayCounter = 0; nArrayCounter < 24;
nArrayCounter++ )
    {
        bool bIsOffsetsEqual    = anPlaceArray [ 1 ][ nRPoffset
] == anRightValues [ nArrayCounter][ nRPoffset    ];
        bool bIsSizesEqual      = anPlaceArray [ 1 ][ nRPsize
] == anRightValues [ nArrayCounter][ nRPsize      ];
        bool bIsStringsEqual    = anPlaceArray [ 1 ][ nRPstring
] == anRightValues [ nArrayCounter][ nRPstring    ];
        bool bIsTypesEqual      = anPlaceArray [ 1 ][ nRPtype
] == anRightValues [ nArrayCounter][ nRPtype      ];
        bool bIsBitsEqual       = anPlaceArray [ 1 ][ nRPbits
] == anRightValues [ nArrayCounter][ nRPbits      ];
        bool bIsGoodValuesEqual = anPlaceArray [ 1 ][
nRPgoodvalue ] == anRightValues [ nArrayCounter][ nRPgoodvalue
];

        if ( bIsOffsetsEqual && bIsSizesEqual && bIsStringsEqual
&& bIsTypesEqual && bIsBitsEqual && bIsGoodValuesEqual )
        {
            bRecordOK = true;
            break;
        }
    }
    return bRecordOK;
}
```

```
void XML_Check::ShowRecordErrors(int(&anPlaceArray)[2][6],
QStringList *aCNames, int nCurrentUnit, int nCurrentRecord,
QStringList * aCElementNamesR, QStringList *aCErrorMessages )
{
    QString CErrorMessage = "";
    try
    {

        for(int nNameCounter = 0; nNameCounter < 6;
nNameCounter++)
        {
            bool bIsParamRepeat   = ( anPlaceArray [ 0 ][
nNameCounter ] == nXPRepeatParametr );
            bool bIsParamNull     = ( anPlaceArray [ 0 ][
nNameCounter ] == nXPEmptyParametr );
            bool bIsParamWrong    = ( anPlaceArray [ 1 ][
nNameCounter ] == nXPEmptyParametr );
            bool bIsParamRequired = ( nNameCounter < nRPbits );

            // Если эл-т повторяется
            if ( bIsParamRepeat )
            {
                CErrorMessage = aCErrorMessages-
>at(EMCModulePropRepeatError) + " (" + aCNames->at(nNameCounter)
+ ").";
                throw new std::exception;
            }
            else if(bIsParamRequired)
            {
                // Отсутствие обязательного эл-та
                if ( bIsParamNull )
                {
                    CErrorMessage = aCErrorMessages-
>at(EMCModulePropNullError) + " (" + aCNames->at(nNameCounter) +
").";
                    throw new std::exception;
                }
                // Неверный формат параметра
                if ( bIsParamWrong )
                {
                    CErrorMessage = aCErrorMessages-
>at(EMModulePropFormatError) + " (" + aCNames->at(nNameCounter)
+ ").";
                    throw new std::exception;
                }
            }

        }

        bool bIsBitsFind      = anPlaceArray [ 1 ][ nRPbits
] == nROVRight;
```

```
        bool bIsGoodvalueFind = anPlaceArray [ 1 ][ nRPgoodvalue
] >  nRGVRecordGoodvalueNotFind;

        // bits и goodvalue в одном блоке
        if ( bIsBitsFind && bIsGoodvalueFind )
        {
            CErrorMessage = aCErrorMessages-
>at(EMRecordBitsGoodvalueError);
            throw new std::exception;
        }

        bool bIsGoodvalueBig   = anPlaceArray [ 1 ][
nRPgoodvalue ] > anPlaceArray [ 1 ][ nRPsize ];
        bool bIsGoodvalueWrong = anPlaceArray [ 1 ][
nRPgoodvalue ] == nRGVGoodvalueWrong;

        // Хорошее значение имеет неверный формат
        if ( bIsGoodvalueBig || bIsGoodvalueWrong )
        {
            CErrorMessage = aCErrorMessages-
>at(EMRecordGoodvalueError);
            throw new std::exception;
        }

        bool bIsType_b = anPlaceArray [ 1 ][ nRPtype ] ==
nRTVType_b;

        // bits в поле не битового типа
        if ( !bIsType_b && bIsBitsFind )
        {
            CErrorMessage = aCErrorMessages-
>at(EMRecordBitsTypeError);
            throw new std::exception;
        }

        // bits нет в поле битового типа
        if ( bIsType_b && !bIsBitsFind )
        {
            CErrorMessage = aCErrorMessages-
>at(EMRecordBitsNullError);
            throw new std::exception;
        }

        bool bIsType_ip     = anPlaceArray [ 1 ][ nRPtype ] ==
nRTVType_IP;
        bool bIsSize_4bytes = anPlaceArray [ 1 ][ nRPsize ] ==
nRSVSize_4byte;

        // Размер не соответствует типу ip
        if ( bIsType_ip && !bIsSize_4bytes )
        {
            CErrorMessage = aCErrorMessages-
>at(EMRecordIpSizeError);
```

```
            throw new std::exception;
        }

        bool bIsType_mac    = anPlaceArray [ 1 ][ nRPtype ] ==
nRTVType_Mac;
        bool bIsSize_6bytes = anPlaceArray [ 1 ][ nRPsize ] ==
nRSVSize_6byte;

        // Размер не соответствует типу mac
        if ( bIsType_mac && !bIsSize_6bytes )
        {
            CErrorMessage = aCErrorMessages-
>at(EMRecordMacSizeError);
            throw new std::exception;
        }

        // Goodvalue в битовом или mac или ip
        if ( ( bIsType_b || bIsType_ip || bIsType_mac ) &&
bIsGoodvalueFind )
        {
            CErrorMessage = aCErrorMessages-
>at(EMGoodvalueInWrongType);
            throw new std::exception;
        }
    }
    catch (...)
    {
        QErrorMessage CErrMsg;
        CErrMsg.showMessage ( CErrorMessage +
                              " (" + QString::number(nCurrentUnit) +
" " + aCElementNamesR->at(nEUnit) + ", " +
                              QString::number(nCurrentRecord) + " "
+ aCElementNamesR->at(nERecord) + ")." +
                              aCErrorMessages-
>at(EMCheckFile),aCErrorMessages->at(EMError));
        CErrMsg.exec();
    }
}

bool XML_Check::CheckBitValue(QDomNode *CBit,
                              int(&anPlaceArray)[5],
                              QStringList *aCNames,
                              QStringList *aCElementNamesR,
                              QStringList aCErrorMessages,
                              QStringList aCTypeVal,
                              int nCurrentUnit,
                              int nCurrentRecord,
                              int nCurrentBit,
                              uint uRecordSize,
                              bool bErrorMessageShow)
{
    QDomNode COffset    = CBit->childNodes().at(nBPoffset);
    QDomNode CSize      = CBit->childNodes().at(nBPsize);
```

```
    QDomNode CType      = CBit->childNodes().at(nBPtype);
    QDomNode CGoodvalue = CBit->childNodes().at(nBPgoodvalue);

    // Признак правильности свойств записи
    bool bBitOK = true;
    // Для парсинга
    uint uParser;

    QString CErrorMessage = "";

    // Цикл проверки
    try
    {

        for ( int nNameCounter = 0; nNameCounter < aCNames-
>length(); nNameCounter++ )
        {
            bool bIsParamRepeat = anPlaceArray [ nNameCounter ]
== nXPRepeatParametr;

            // Эл-т повторяется
            if ( bIsParamRepeat )
            {
                CErrorMessage =
aCErrorMessages.at(EMCModulePropRepeatError) + " (" + aCNames-
>at(nNameCounter) + ").";
                throw new std::exception;
            }

            bool bIsParamExists     = anPlaceArray [
nNameCounter ] > nXPEmptyParametr;
            bool bIsGoodvalueExists = anPlaceArray [
nBPgoodvalue ] > nXPEmptyParametr;
            bool bNoInnerXml        = bIsParamExists && ( CBit-
>childNodes().at( nNameCounter ).childNodes().count() > 0 &&
                                                          CBit-
>childNodes().at( nNameCounter ).childNodes().at( 0 ).nodeType()
!= QDomNode::TextNode );

            bool bIsSizeExists      = anPlaceArray [ nBPsize ] >
nXPEmptyParametr;

            bool bIsGoodvalueWrong  = bIsGoodvalueExists &&
bIsSizeExists && !ValueIsValid (
CGoodvalue.childNodes().at(0).nodeValue(),

(int)CSize.childNodes().at(0).nodeValue().toUInt(&bIsGoodvalueWr
ong,10) );
            bool bIsParamRequired   = nNameCounter !=
nBPgoodvalue;

            // Обязательные эл-ты
            if ( bIsParamRequired )
```

```
            {
                bool bIsParamNull = anPlaceArray [ nNameCounter
] == nXPEmptyParametr;

                // Эл-т отсутствует
                if ( bIsParamNull )
                {
                    CErrorMessage =
aCErrorMessages.at(EMCModulePropNullError) + " (" + aCNames-
>at(nNameCounter) + ").";
                    throw new std::exception;
                }

                else
                {
                    bool bIsType = ( nNameCounter == nBPtype );
                    bool bIsTypeBad = bIsType && !(
CType.childNodes().at(0).nodeValue() == aCTypeVal [ VType_D ] ||

CType.childNodes().at(0).nodeValue() == aCTypeVal [ VType_X ] ||

CType.childNodes().at(0).nodeValue() == aCTypeVal [ VType_B ] );
                    bool bIsUint;
                    uParser = CBit->childNodes().at(
anPlaceArray [ nNameCounter ]
).childNodes().at(0).nodeValue().toUInt(&bIsUint,10);

                    bIsUint = (bIsUint == true) && (uParser <
INT_MAX );

                    bool bIsOffsetOrSize = nNameCounter <
nBPstring;

                    // Если параметр смещения, размера и типа
имеет неверное значение
                    if ( bIsTypeBad || ( bNoInnerXml || (
bIsOffsetOrSize && !bIsUint ) ) )
                    {
                        CErrorMessage =
aCErrorMessages.at(EMModulePropFormatError) + " (" + aCNames-
>at(nNameCounter) + ").";
                        throw new std::exception;
                    }
                    else
                    {

                        bool bIsSizeUint;
                        uParser =
CSize.childNodes().at(0).nodeValue().toUInt(&bIsSizeUint,10);

                        bIsSizeUint = ( bIsSizeUint == true ) &&
uParser < INT_MAX;
```

```
                        bool bIsBitBiggerRecord;
                        uint uOffset =
COffset.childNodes().at(0).nodeValue().toUInt(&bIsBitBiggerRecor
d);
                        uint uSize   =
CSize.childNodes().at(0).nodeValue().toUInt(&bIsBitBiggerRecord)
;

                        bIsBitBiggerRecord = bIsBitBiggerRecord
&& bIsSizeUint && ( uOffset + uSize > uRecordSize * 8 );
                        // Если бит больше записи
                        if ( bIsSizeExists && bIsBitBiggerRecord
)
                        {
                            CErrorMessage =
aCErrorMessages.at(EMBitOffsetSizeError);
                            throw new std::exception;
                        }

                    }
                }
            }

            // Если goodvalue есть, то оно не соответствует
размеру бита или имеет неверный формат
            else if ( bIsGoodvalueExists && ( bNoInnerXml ||
bIsGoodvalueWrong ) )
            {
                CErrorMessage =
aCErrorMessages.at(EMRecordGoodvalueError);
                throw new std::exception;
            }


        }
    } catch (...)
    {
        // Флаг ошибки
        bBitOK = false;

        // Сообщение об ошибке
        if ( bErrorMessageShow )
        {
            QErrorMessage CErrMsg;
            CErrMsg.showMessage(CErrorMessage + " (" +
                                QString::number(nCurrentUnit) + " "
+ aCElementNamesR->at(nEUnit) + ", " +
                                QString::number(nCurrentRecord) + "
" + aCElementNamesR->at(nERecord) + ", " +
                                QString::number(nCurrentBit) + " " +
aCElementNamesR->at(nEBit) + ")." +
```

```
aCErrorMessages.at(EMCheckFile),aCErrorMessages.at(EMError));
            CErrMsg.exec();
        }
    }

    return bBitOK;
}

bool XML_Check::CheckQDocument(QDomDocument *CQDoc,
QList<QStringList> *aCNames, QStringList
*aCElementNamesE,QStringList *aCElementNamesR, QStringList
aCErrorMessages, QStringList aCTypeVal)
{
    bool bIsDocOK = true;
    QDomElement CRoot = CQDoc->documentElement();
    QDomElement *CPRoot = &CRoot;
    if(CPRoot == nullptr || CQDoc->documentElement().tagName()
!= aCElementNamesE->at(nEModel))
    {
        bIsDocOK = false;
        // Вставить сюда сообщение об ошибке
    }
    else
        bIsDocOK = CheckUnits(CPRoot, aCNames,
aCElementNamesE,aCElementNamesR,aCErrorMessages,aCTypeVal);
    return  bIsDocOK;
}

bool XML_Check::CheckBits(QDomNode *CBits,
                          QList<QStringList> *aCNames,
                          QStringList *aCElementNamesE,
                          QStringList *aCElementNamesR,
                          QStringList aCErrorMessages,
                          QStringList aCTypeVal,
                          int nCurrentUnit,
                          int nCurrentRecord,
                          uint uRecordSize,
                          bool bErrorMessageShow)
{
    // Признак правильности свойств bits
    bool bBitsOK = true;
    // Обнуляем счётчик бит
    int nCurrentBit = 0;

    // Проход по списку эл-тов bits
    for ( int nChildCounter = 0; nChildCounter < CBits-
>childNodes().count(); nChildCounter++ )
    {
        bool bIsNameBit = CBits->childNodes().at( nChildCounter
).nodeName() == aCElementNamesE->at(nEBit);
        // Если называется "bit", то проверяем его правильность
        if ( bIsNameBit )
```

```
        {
            QDomNode CBit = CBits->childNodes().at(
nChildCounter );
            // Увеличиваем счётчик бит
            nCurrentBit++;
            // Проверяем бит
            bBitsOK =
CheckBit(&CBit,aCNames,aCElementNamesR,aCErrorMessages,aCTypeVal
,nCurrentUnit,nCurrentRecord,nCurrentBit,uRecordSize,bErrorMessa
geShow);
            if(!bBitsOK) break;
        }

    }
    return bBitsOK;
}

bool XML_Check::CheckBit(QDomNode *CBit,
                         QList<QStringList> *aCNames,
                         QStringList *aCElementNamesR,
                         QStringList aCErrorMessages,
                         QStringList aCTypeVal,
                         int nCurrentUnit,
                         int nCurrentRecord,
                         int nCurrentBit,
                         uint uRecordSize,
                         bool bErrorMessageShow)
{
    QStringList aCBitNames = aCNames->at(nEBit);
    // Признак правильности свойств бита
    bool bBitOK = true;
    // Массив значений и положений эл-тов биту
    int anPlaceArray[nBPgoodvalue + 1] =
{nXPEmptyParametr,nXPEmptyParametr,nXPEmptyParametr,nXPEmptyPara
metr,nXPEmptyParametr};

    // Проход по списку эл-тов bit
    for ( int nChildCounter = 0; nChildCounter < CBit-
>childNodes().count(); nChildCounter++ )
    {
        // Проход по массиву допустимых названий
        for ( int nNameCounter = 0; nNameCounter <
aCBitNames.length(); nNameCounter++ )
        {
            bool bNameIsParam = CBit-
>childNodes().item(nChildCounter).nodeName() == aCBitNames.at(
nNameCounter );
            // Если не соответствует названию, продолжить
выполнение цикла
            if ( !bNameIsParam ) continue;

            bool bIsParamNotRepeat = anPlaceArray [ nNameCounter
] == nXPEmptyParametr;
```

```
            // Если эл-т не найден ранее, записываем положение,
если повторяется, ставим признак повтора
            if ( bIsParamNotRepeat )
                anPlaceArray [ nNameCounter ] = nChildCounter;
            else anPlaceArray [ nNameCounter ] =
nXPRepeatParametr;
        }
    }
    bBitOK =
CheckBitValue(CBit,anPlaceArray,&aCBitNames,aCElementNamesR,aCEr
rorMessages,aCTypeVal,nCurrentUnit,nCurrentRecord,nCurrentBit,uR
ecordSize,bErrorMessageShow);
    return bBitOK;
}

bool XML_Check::ValueIsValid(QString CValue, int32_t nP)
{
    // Признак правильности
    bool bResVal = false;
    // Если признак меньше или равен 32
    if ( (nP > 0) && (nP <= 32) )
    {
        // Максимальное значение
        unsigned long long dwMaxValue = 1;
        // Расчитываем максимальное значение, сдвигая его влево
nP раз
        dwMaxValue = dwMaxValue << nP;
        // Переменная для записи конвертированной строки
        unsigned long long dwCurrentValue = ULONG_MAX;
        if(CValue.length() > 2 && CValue.at(0) == '0' &&
CValue.at(1) == 'x')
            dwCurrentValue = CValue.toULongLong(&bResVal,16);
        else
            dwCurrentValue = CValue.toULongLong(&bResVal,10);

        if(bResVal && dwCurrentValue >= dwMaxValue)
            bResVal = false;
    }
    return bResVal;
}

QDomNode XML_Check::FindChildXml(QString CName, QDomNode
CParentXML)
{
    QDomNode CFindXml;
    for(int i = 0; i < CParentXML.childNodes().count(); i++)
    {
        if (CParentXML.childNodes().at(i).nodeName() == CName)
        {
            CFindXml = CParentXML.childNodes().at(i);
            break;
        }
    }
```

```
    return CFindXml;
}

void XML_Check::CheckUnitsCount(QDomDocument *CXDoc,
QList<QStringList> *aCNames, QStringList *acElementNamesE )
{
    int nCurrentUnitCount = 0;
    int nCountUnitPosition = -1;

    for(int i = 0; i < CXDoc-
>documentElement().childNodes().count(); i++)
    {
        if(CXDoc-
>documentElement().childNodes().at(i).nodeName() ==
acElementNamesE->at(nEUnit))
            nCurrentUnitCount++;

        if(CXDoc-
>documentElement().childNodes().at(i).nodeName() == aCNames-
>at(nEModel).at(nMPCountUnit))
            nCountUnitPosition = i;

    }

    CXDoc-
>documentElement().childNodes().at(nCountUnitPosition).childNode
s().at(0).setNodeValue(QString::number(nCurrentUnitCount));
}

void XML_Check::CalcMaxSizeRecord(QDomDocument *CXDoc,
QList<QStringList> *aCNames, QStringList *acElementNamesE )
{
    QDomNode CRoot = CXDoc->documentElement();
    int nMaxSizeRecordPostition = -1;
    int nMaxSize = 0;
    for(int i = 0; i < CRoot.childNodes().count(); i++)
    {
        if(CRoot.childNodes().at(i).nodeName() == aCNames-
>at(nEModel).at(nMPMaxSizeRecord))
            nMaxSizeRecordPostition = i;
        else if(CRoot.childNodes().at(i).nodeName() ==
acElementNamesE->at(nEUnit))
        {
            QDomNode CUnit = CRoot.childNodes().at(i);
            QDomNode CRecords = FindChildXml(aCNames-
>at(nEUnit).at(nUPrecords),CUnit);
            for(int j = 0; j < CRecords.childNodes().count();
j++)
            {
                if(CRecords.childNodes().at(j).nodeName() !=
acElementNamesE->at(nERecord)) continue;
                QDomNode CRecord = CRecords.childNodes().at(j);
```

```
                QDomNode CSize = FindChildXml(aCNames-
>at(nERecord).at(nRPsize), CRecord);
                int nSize =
CSize.childNodes().at(0).nodeValue().toInt();
                if(nSize > nMaxSize)
                    nMaxSize = nSize;
            }
        }
    }

CRoot.childNodes().at(nMaxSizeRecordPostition).childNodes().at(0
).setNodeValue(QString::number(nMaxSize));
}

QDomNode XML_Check::CreateModel(QDomDocument *CXDoc,
QList<QStringList> *aCNames, QStringList *acElementNamesE )
{
    QDomNode CUnits = CXDoc->createElement(acElementNamesE-
>at(nEModel));

    QDomNode CCountByteHeader = CXDoc->createElement(aCNames-
>at(nEModel).at(nMPCountByteHeader));
    QDomNode CCountByteHeaderValue = CXDoc->createTextNode("0");
    CCountByteHeader.appendChild(CCountByteHeaderValue);

    QDomNode CCountUnit = CXDoc->createElement(aCNames-
>at(nEModel).at(nMPCountUnit));
    QDomNode CCountUnitValue = CXDoc->createTextNode("0");
    CCountUnit.appendChild(CCountUnitValue);

    QDomNode CMaxSizeRecord = CXDoc->createElement(aCNames-
>at(nEModel).at(nMPMaxSizeRecord));
    QDomNode CMaxSizeRecordValue = CXDoc->createTextNode("0");
    CMaxSizeRecord.appendChild(CMaxSizeRecordValue);

    CUnits.appendChild(CCountByteHeader);
    CUnits.appendChild(CCountUnit);
    CUnits.appendChild(CMaxSizeRecord);

    return CUnits;
}

QDomNode XML_Check::CreateUnit(QDomDocument *CXDoc,
QList<QStringList> *aCNames, QStringList *acElementNamesE )
{
    QDomNode CUnit = CXDoc->createElement(acElementNamesE-
>at(nEUnit));

    QDomNode CUnitCodeOperation = CXDoc->createElement(aCNames-
>at(nEUnit).at(nUPcodeOperation));
    QDomNode CUnitCodeOperationValue = CXDoc-
>createTextNode("0");
    CUnitCodeOperation.appendChild(CUnitCodeOperationValue);
```

```
    QDomNode CUnitInformation = CXDoc->createElement(aCNames-
>at(nEUnit).at(nUPinformation));
    QDomNode CUnitInformationValue = CXDoc->createTextNode("");
    CUnitInformation.appendChild(CUnitInformationValue);

    QDomNode CUnitRecords = CXDoc->createElement(aCNames-
>at(nEUnit).at(nUPrecords));
    CUnitInformation.appendChild(CUnitRecords);

    CUnit.appendChild(CUnitCodeOperation);
    CUnit.appendChild(CUnitInformation);
    CUnit.appendChild(CUnitRecords);

    return CUnit;
}

QDomNode XML_Check::CreateRecord(QDomDocument *CXDoc,
QList<QStringList> *aCNames, QStringList *acElementNamesE,
QStringList *aCTypeVal)
{
    QDomNode CRecord = CXDoc->createElement(acElementNamesE-
>at(nERecord));

    QDomNode CRecordOffset = CXDoc->createElement(aCNames-
>at(nERecord).at(nRPoffset));
    QDomNode CRecordOffsetValue = CXDoc->createTextNode("0");
    CRecordOffset.appendChild(CRecordOffsetValue);

    QDomNode CRecordSize = CXDoc->createElement(aCNames-
>at(nERecord).at(nRPsize));
    QDomNode CRecordSizeValue = CXDoc->createTextNode("0");
    CRecordSize.appendChild(CRecordSizeValue);

    QDomNode CRecordString = CXDoc->createElement(aCNames-
>at(nERecord).at(nRPstring));
    QDomNode CRecordStringValue = CXDoc->createTextNode("");
    CRecordString.appendChild(CRecordStringValue);

    QDomNode CRecordType = CXDoc->createElement(aCNames-
>at(nERecord).at(nRPtype));
    QDomNode CRecordTypeValue = CXDoc->createTextNode(aCTypeVal-
>at(VType_D));
    CRecordType.appendChild(CRecordTypeValue);

    CRecord.appendChild(CRecordOffset);
    CRecord.appendChild(CRecordSize);
    CRecord.appendChild(CRecordString);
    CRecord.appendChild(CRecordType);

    return CRecord;
}
```

```
QDomNode XML_Check::CreateBit(QDomDocument *CXDoc,
QList<QStringList> *aCNames, QStringList *acElementNamesE,
QStringList *aCTypeVal)
{
    QDomNode CBit = CXDoc->createElement(acElementNamesE-
>at(nEBit));

    QDomNode CBitOffset = CXDoc->createElement(aCNames-
>at(nEBit).at(nBPoffset));
    QDomNode CBitOffsetValue = CXDoc->createTextNode("0");
    CBitOffset.appendChild(CBitOffsetValue);

    QDomNode CBitSize = CXDoc->createElement(aCNames-
>at(nEBit).at(nBPsize));
    QDomNode CBitSizeValue = CXDoc->createTextNode("0");
    CBitSize.appendChild(CBitSizeValue);

    QDomNode CBitString = CXDoc->createElement(aCNames-
>at(nEBit).at(nBPstring));
    QDomNode CBitStringValue = CXDoc->createTextNode("");
    CBitString.appendChild(CBitStringValue);

    QDomNode CBitType = CXDoc->createElement(aCNames-
>at(nEBit).at(nBPtype));
    QDomNode CBitTypeValue = CXDoc->createTextNode(aCTypeVal-
>at(VType_D));
    CBitType.appendChild(CBitTypeValue);

    CBit.appendChild(CBitOffset);
    CBit.appendChild(CBitSize);
    CBit.appendChild(CBitString);
    CBit.appendChild(CBitType);

    return CBit;
}
```

Приложение А. 9 – файл «xml_editor.cpp»

```
#include "xml_editor.h"
#include "ui_xml_editor.h"



XML_Editor::XML_Editor(QWidget *parent) : QMainWindow(parent),
ui(new Ui::XML_Editor)
{
    ui->setupUi(this);
    tableTree = new TableTree();
    ui->horizontalLayout->addWidget(ui->treeWidget);
    ui->horizontalLayout->addSpacing(5);
    ui->horizontalLayout->addWidget(tableTree);

connect(tableTree,SIGNAL(checkIsEnd(bool,bool)),this,SLOT(EndOfC
heckTable(bool,bool)));
```

```
    ui->treeWidget-
>setContextMenuPolicy(Qt::ContextMenuPolicy::CustomContextMenu);
    connect(ui-
>treeWidget,SIGNAL(customContextMenuRequested(const
QPoint&)),this,SLOT(ShowContextMenu(const QPoint&)));

    ui->centralwidget->setLayout(ui->verticalLayout);
    SetupMenu();
}

XML_Editor::~XML_Editor()
{
    delete ui;
}

void XML_Editor::SetupMenu()
{
    m_CMenu.addActions(m_aCActions);


connect(m_aCActions.at(A_AddChild),SIGNAL(triggered()),this,SLOT
(on_pushButton_4_clicked()));

connect(m_aCActions.at(A_Remove),SIGNAL(triggered()),this,SLOT(o
n_pushButton_5_clicked()));

connect(m_aCActions.at(A_AddHigher),SIGNAL(triggered()),this,SLO
T(AddElementHigher()));

connect(m_aCActions.at(A_AddLower),SIGNAL(triggered()),this,SLOT
(AddElementLower()));

connect(m_aCActions.at(A_ReplaceHigher),SIGNAL(triggered()),this
,SLOT(ReplaceElementHigher()));

connect(m_aCActions.at(A_ReplaceLower),SIGNAL(triggered()),this,
SLOT(ReplaceElementLower()));
}

int XML_Editor::GetNodeLevel(QTreeWidgetItem *CItem)
{
    QTreeWidgetItem *CCurrentItem = CItem;
    int nCurrentLevel = 0;
    while (CCurrentItem->parent()!=nullptr)
    {
        nCurrentLevel++;
        CCurrentItem = CCurrentItem->parent();
    }
    return nCurrentLevel;
}

void XML_Editor::on_action_5_triggered()
{
```

```
    this->close();
}



void XML_Editor::on_pushButton_3_clicked()
{
    QString filename = "";
    filename =
QFileDialog::getOpenFileName(this,QString::fromUtf8("Загрузить
файл"),QDir::currentPath(),"XML (*.xml)");
    if(filename != "")
    {
        QFile file(filename);
        QDomDocument *CTempDoc = new QDomDocument();
        CTempDoc->setContent(&file);


if(XML_Check::CheckQDocument(CTempDoc,&g_aaCNamesString,&g_aCEle
mentNamesE,&g_aCElementNamesR,g_aCErrorMessages,g_aCTypeVal))
        {
            g_CXDoc = CTempDoc;

UpdateTree(g_CXDoc,&g_aCElementNamesE,&g_aCElementNamesR,&g_aaCN
amesString);
            ui->treeWidget->expandAll();
            tableTree->ClearTable();
            g_CSelectedTreeItem = nullptr;
            g_CCurrentParentNode.clear();
        }
        else
        {
            QErrorMessage msg;

msg.showMessage(g_aCErrorMessages.at(EMLoadFileFailure),g_aCErro
rMessages.at(EMError));
            msg.exec();
        }
    }
}

void XML_Editor::UpdateTree(QDomDocument* CXDoc, QStringList*
aCElementNamesE, QStringList* aCElementNamesR,
QList<QStringList> *aaCNamesString)
{
    ui->treeWidget->blockSignals(true);
    // Ссылка на корневой элемент дерева
    QTreeWidgetItem *CRoot;
    // Если корневой элемент пустой, то создать, если нет, то
очистить потомков
    if(ui->treeWidget->topLevelItemCount() == 0)
    {
        CRoot = new QTreeWidgetItem();
```

```
        CRoot->setText(0,aCElementNamesR->at(nEModel));
        ui->treeWidget->addTopLevelItem(CRoot);
    }
    else
    {
        CRoot = ui->treeWidget->topLevelItem(0);

        // Проблемный цикл
        for(int i = 0; i < CRoot->childCount(); i++)
        {
            QTreeWidgetItem *CChild = CRoot->child(i);
            int nChildIndex = CRoot->indexOfChild(CChild);
            CRoot->removeChild(CRoot->child(nChildIndex));
            i--;
        }
    }

    CRoot = ui->treeWidget->topLevelItem(0);
    // Текущий элемент XML

    for(int i = 0; i < CXDoc-
>documentElement().childNodes().count(); i++)
    {
        QDomNode CCurrentXmlParam = CXDoc-
>documentElement().childNodes().at(i);
        bool bIsUnit = ( CXDoc-
>documentElement().childNodes().at(i).nodeName() ==
aCElementNamesE->at(nEUnit ));

        // Если потомок корневого эл-та называется Unit
        if ( !bIsUnit ) continue;

        // Добавить эл-т в дерево
        QTreeWidgetItem *CUnit = new QTreeWidgetItem();
        CUnit->setText(0,aCElementNamesR->at(nEUnit ));
        CRoot->addChild(CUnit);

        // Создать модули в дереве

UpdateTreeUnitsRecords(CUnit,CCurrentXmlParam,true,aCElementName
sE,aCElementNamesR, aaCNamesString);
    }
    ui->treeWidget->expandAll();
    ui->treeWidget->clearSelection();
    ui->treeWidget->setCurrentItem(nullptr);
    ui->treeWidget->blockSignals(false);
}

void XML_Editor::UpdateTreeUnitsRecords(QTreeWidgetItem
*CCurrentItem, QDomNode CParentXmlNode, bool bIsUnit,
QStringList* aCElementNamesE, QStringList* aCElementNamesR,
QList<QStringList> *aaCNamesString)
{
```

```
    // Определяем название роидтельского компонента в дереве и
название свойства, содержащего его имя,
    // имя списка дочерних элементов, имя дочернего компонента в
дереве и XML

    QString CParentName, CParentXmlName, CListName,
CChildXmlName, CChildTreeName;

    // Если родительский элемент - модуль

    if ( bIsUnit )
    {
        CParentName    = aCElementNamesR->at(nEUnit);
// Модуль
        CParentXmlName = aaCNamesString-
>at(nEUnit).at(nUPinformation); // information
        CListName      = aaCNamesString-
>at(nEUnit).at(nUPrecords);     // records
        CChildXmlName  = aCElementNamesE->at(nERecord);
// record
        CChildTreeName = aCElementNamesR->at(nERecord);
// Запись
    }
    else
    {
        CParentName    = aCElementNamesR->at(nERecord);
// Запись
        CParentXmlName = aaCNamesString-
>at(nERecord).at(nRPstring); // string
        CListName      = aaCNamesString-
>at(nERecord).at(nRPbits);   // bits
        CChildXmlName  = aCElementNamesE->at(nEBit);
// bit
        CChildTreeName = aCElementNamesR->at(nEBit);
// Бит
    }

    // Цикл по количеству потомков родительского XML

    for ( int i = 0; i < CParentXmlNode.childNodes().count();
i++ )
    {
        // Если потомок носит имя эл-та, хранящего имя родителя,
до добавить имя родительскому компоненту
        bool bIsNameParent =
CParentXmlNode.childNodes().at(i).nodeName() == CParentXmlName;

        if ( bIsNameParent )
        {
            QString CNewName = CCurrentItem->text(0) + " (" +
CParentXmlNode.childNodes().at(i).childNodes().at(0).nodeValue()
+ ")";
            CCurrentItem->setText(0,CNewName);
```

```
            continue;
        }
        // Если потомок не носит имя списка дочерних XML-
компонентов
        bool bIsNameChildList =
CParentXmlNode.childNodes().at(i).nodeName() == CListName;
        if ( !bIsNameChildList ) continue;

        // Цикл по списку дочерних компонентов списка
        for ( int j = 0; j <
CParentXmlNode.childNodes().at(i).childNodes().count(); j++ )
        {
            // Если имя эл-та равняется имени потомка
            bool bIsNameChild =
CParentXmlNode.childNodes().at(i).childNodes().at(j).nodeName()
== CChildXmlName;
            if ( !bIsNameChild ) continue;

            // Добавить эл-т в дерево
            QTreeWidgetItem *CChild = new QTreeWidgetItem();
            CChild->setText(0,CChildTreeName);
            CCurrentItem->addChild(CChild);

            // Если имя "record", то вызвать эту же функцию с
параметрами для record, иначе вызвать функцию добавления бит
            bool bIsNameRecord = CChildXmlName ==
aCElementNamesE->at(nERecord);

            if ( bIsNameRecord )
            {
                QDomNode CRecord =
CParentXmlNode.childNodes().at(i).childNodes().at(j);

UpdateTreeUnitsRecords(CChild,CRecord,false,aCElementNamesE,aCEl
ementNamesR,aaCNamesString);
            }

            else
            {
                QDomNode CBit =
CParentXmlNode.childNodes().at(i).childNodes().at(j);
                UpdateTreeBits(CChild,CBit,aaCNamesString-
>at(nEBit).at(nRPstring),CChildTreeName);
            }

        }

    }
}

void XML_Editor::UpdateTreeBits(QTreeWidgetItem *CCurrentItem,
QDomNode CXmlBit, QString CXStringName, QString CBitName)
{
```

```
    // Цикл по кол-ву эл-тов XML
    for ( int i = 0; i < CXmlBit.childNodes().count(); i++ )
    {
        // Если эл-т хранит не название бита, то продолжить
цикл, иначе внести изменение в его имя
        bool bIsNameString = CXmlBit.childNodes().at( i
).nodeName() == CXStringName;

        if ( !bIsNameString ) continue;

        CCurrentItem->setText(0,CBitName + " (" +
CXmlBit.childNodes().at(i).childNodes().at(0).nodeValue() +
")");
    }
}

QDomNode XML_Editor::FindXml(QDomDocument *CXDoc,QTreeWidgetItem
*CSelectedNode, QList<QStringList> *aaCNamesString, QStringList
*aCElementNamesE)
{
    // Текущий компонент XML
    QDomNode CCurrentXML = CXDoc->documentElement();

    // Текущий эл-т дерева
    QTreeWidgetItem *CCurrentTreeNode;

    int nLevelOfSelectedNode = GetNodeLevel(CSelectedNode);

    for(int i = 0; i < nLevelOfSelectedNode; i++)
    {
        // Текущему компоненту дерева присвоить выбранный
        CCurrentTreeNode = CSelectedNode;

        for ( int j = 0; j < nLevelOfSelectedNode - i - 1; j++ )
            // Поиск родительского эл-та дерева
            CCurrentTreeNode = CCurrentTreeNode->parent();

        int nLevelOfCurrentNode =
GetNodeLevel(CCurrentTreeNode);
        bool bIsDocument = CCurrentXML == CXDoc-
>documentElement();
        bool bIsRecord = nLevelOfCurrentNode == nERecord;

        // Поиск модуля
        if ( bIsDocument )
            // Функция поиска модуля
            CCurrentXML =
FindUnit(&CCurrentXML,CCurrentTreeNode->parent()-
>indexOfChild(CCurrentTreeNode),aCElementNamesE);

        // Поиск записи
        else if ( bIsRecord )
            // Функция поиска записи
```

```
            CCurrentXML =
FindRecordOrBit(&CCurrentXML,CCurrentTreeNode->parent()-
>indexOfChild(CCurrentTreeNode),aaCNamesString-
>at(nEUnit).at(nUPrecords),aCElementNamesE->at(nERecord));

        // Поиск бита
        else
            CCurrentXML =
FindRecordOrBit(&CCurrentXML,CCurrentTreeNode->parent()-
>indexOfChild(CCurrentTreeNode),aaCNamesString-
>at(nERecord).at(nRPbits),aCElementNamesE->at(nEBit));

    }
    return CCurrentXML;
}

QDomNode XML_Editor::FindUnit(QDomNode *CCurrentXML,int
nCCurrentNodeIndex, QStringList *aCElementNamesE)
{
    // Устанавливаем начальное значение счётчика модулей
    int nUnitCounter = -1;

    QDomNode CFindXML;
    // Цикл по кол-ву эл-тов XML
    for ( int j = 0; j < CCurrentXML->childNodes().count(); j++
)
    {
        // Если элемент называется Unit, то увеличиваем счётчик
модулей и сравниваем его с индексом эл-та

        bool bIsUnit = CCurrentXML-
>childNodes().at(j).nodeName() == aCElementNamesE->at(nEUnit);
        if ( !bIsUnit ) continue;

        nUnitCounter++;

        // Если индекс равен счётчику, выходим из цикла
        bool bIsIndex = nUnitCounter == nCCurrentNodeIndex;
        if ( !bIsIndex ) continue;

        CFindXML = CCurrentXML->childNodes().at(j);
        // CCurrentXML = &CFindXML;

    }
    return CFindXML;
}

QDomNode XML_Editor::FindRecordOrBit(QDomNode *CCurrentXML,int
nCCurrentNodeIndex, QString CContainerName, QString CXmlName)
{
    QDomNode CFindXml;
    // Цикл по кол-ву потомков
```

```
    for ( int j = 0; j < CCurrentXML->childNodes().count(); j++
)
    {
        // Эл-т является эл-том, хранящим потомков
        bool bIsContainer = CCurrentXML-
>childNodes().at(j).nodeName() == CContainerName;
        if ( !bIsContainer ) continue;

        // Начальное значение счётчика
        int nCounter = -1;

        // Цикл по кол-ву потомков контейнера
        for ( int k = 0; k < CCurrentXML-
>childNodes().at(j).childNodes().count(); k++ )
        {
            // Если эл-т называется как потомок, то увеличиваем
счётчик и если индекс совпал со счётчиком, выходим из цикла
            bool bIsElementName = CCurrentXML-
>childNodes().at(j).childNodes().at(k).nodeName() == CXmlName;

            if ( bIsElementName )
                nCounter++;

            bool bIsIndexesEqual = nCounter ==
nCCurrentNodeIndex;
            if ( !bIsIndexesEqual ) continue;

            CFindXml = CCurrentXML-
>childNodes().at(j).childNodes().at(k);
            break;
        }
        break;
    }

    return CFindXml;
}

QList<QDomNode> *XML_Editor::CreateParamList(QList<QDomNode>
*aCValues, QDomNode CParentXmlNode, QString CSNodeName,
QList<QStringList> *aaCNamesString, QStringList
*aCElementNamesE)
{
    // Очищаем список
    aCValues->clear();

    int nLevelOfNode = 0;
    for(nLevelOfNode = 0; nLevelOfNode < aCElementNamesE-
>count(); nLevelOfNode++)
    {
        if(CSNodeName == aCElementNamesE->at(nLevelOfNode))
            break;
    }
```

```
    QStringList aCNames = aaCNamesString->at(nLevelOfNode);

    // Эл-ты нужны в упорядоченном виде, поэтому сперва цикл по
кол-ву имён
    for (int i = 0; i < aCNames.length(); i++)
    {
        bool bIsNameRecords = aCNames [ i ] == aaCNamesString-
>at(nEUnit).at(nUPrecords);
        bool bIsNameBits    = aCNames [ i ] == aaCNamesString-
>at(nERecord).at(nRPbits);

        // Если свойство необходимо для отображения и
редактирования в таблице
        if ( bIsNameRecords || bIsNameBits ) continue;

        // Цикл по кол-ву потомков XML
        for ( int j = 0; j <
CParentXmlNode.childNodes().count(); j++ )
        {
            // Если имя эл-та XML равно имени искомого эл-та, то
добавляем его в список и выходим из цикла по кол-ву потомков XML
            bool bIsParam = CParentXmlNode.childNodes().at( j
).nodeName() == aCNames [ i ];
            if ( !bIsParam ) continue;

            aCValues->append(CParentXmlNode.childNodes().at(j));
            break;
        }
    }

    return aCValues;
}

void
XML_Editor::on_treeWidget_currentItemChanged(QTreeWidgetItem
*current, QTreeWidgetItem *previous)
{
    ui->treeWidget->blockSignals(true);
    g_CSelectedTreeItem = current;
    // Обновить выбранный элемент XML
    g_CCurrentParentNode =
FindXml(g_CXDoc,g_CSelectedTreeItem,&g_aaCNamesString,&g_aCEleme
ntNamesE);
    int nLevel = GetNodeLevel(g_CSelectedTreeItem);
    g_aCXmlParams =
CreateParamList(g_aCXmlParams,g_CCurrentParentNode,g_CCurrentPar
entNode.nodeName(),&g_aaCNamesString,&g_aCElementNamesE);
    //int nListCount = g_aCXmlParams->count();

    tableTree->blockSignals(true);
    tableTree-
>UpdateTable(g_aCXmlParams,g_CCurrentParentNode.nodeName(),&g_aC
ElementNamesE);
```

```
    tableTree->blockSignals(false);


//UpdateTree(g_CXDoc,&g_aCElementNamesE,&g_aCElementNamesR,&g_aa
CNamesString);
    ui->treeWidget->blockSignals(false);

//XmlEditorUpdateTable(&g_aCTypeVal,&g_aaCNamesString,g_CCurrent
ParentNode,g_CSelectedTreeItem,g_aCXmlParams,&g_aCElementNamesR)
;
}

void::XML_Editor::EndOfCheckTable(bool bNeedUpdateTree, bool
bNeedToUpdateMaxSizeRecord)
{
    tableTree->blockSignals(true);
    if(bNeedUpdateTree)
    {
        ui->treeWidget->blockSignals(true);

UpdateTree(g_CXDoc,&g_aCElementNamesE,&g_aCElementNamesR,&g_aaCN
amesString);
        ui->treeWidget->blockSignals(false);
    }
    if(bNeedToUpdateMaxSizeRecord)

XML_Check::CalcMaxSizeRecord(g_CXDoc,&g_aaCNamesString,&g_aCElem
entNamesE);
    g_aCXmlParams =
CreateParamList(g_aCXmlParams,g_CCurrentParentNode,g_CCurrentPar
entNode.nodeName(),&g_aaCNamesString,&g_aCElementNamesE);
    tableTree-
>UpdateTable(g_aCXmlParams,g_CCurrentParentNode.nodeName(),&g_aC
ElementNamesE);
    tableTree->blockSignals(false);
}




void XML_Editor::on_pushButton_2_clicked()
{
    try
    {

ClearDocument(g_CXDoc,&g_aCElementNamesE,&g_aaCNamesString);

        QString filename =
QFileDialog::getSaveFileName(this,QString::fromUtf8("Сохранить
файл"),QDir::currentPath(),"XML (*.xml)");
        QFile file(filename);
        if(file.open(QIODevice::WriteOnly))
        {
```

```
            QTextStream stream(&file);
            stream<<g_CXDoc->toString();
        }
        file.close();

    }
    catch (...)
    {
        QErrorMessage msg;

msg.showMessage(g_aCErrorMessages.at(EMSaveFileError),g_aCErrorM
essages.at(EMError));
        msg.exec();
    }
}

void XML_Editor::on_checkBox_stateChanged(int arg1)
{
    if(arg1 == Qt::CheckState::Unchecked)
        g_bAllowDialogWindow = false;
    else g_bAllowDialogWindow = true;
}



void XML_Editor::on_pushButton_4_clicked()
{
    if(!g_CCurrentParentNode.isNull())
    {
        bool bNeedToUpdateTree = true;
        bool bNeedToUpdateTable = false;
        QDomNode CChild;
        if(g_CCurrentParentNode.nodeName() ==
g_aCElementNamesE.at(nEModel))
        {
            CChild =
XML_Check::CreateUnit(g_CXDoc,&g_aaCNamesString,&g_aCElementName
sE);
            g_CCurrentParentNode.appendChild(CChild);

XML_Check::CheckUnitsCount(g_CXDoc,&g_aaCNamesString,&g_aCElemen
tNamesE);
            bNeedToUpdateTable = true;
        }
        else if (g_CCurrentParentNode.nodeName() ==
g_aCElementNamesE.at(nEUnit))
        {
            CChild =
XML_Check::CreateRecord(g_CXDoc,&g_aaCNamesString,&g_aCElementNa
mesE,&g_aCTypeVal);
```

```
            QDomNode CRecords =
XML_Check::FindChildXml(g_aaCNamesString.at(nEUnit).at(nUPrecord
s),g_CCurrentParentNode);
            CRecords.appendChild(CChild);

        }
        else if (g_CCurrentParentNode.nodeName() ==
g_aCElementNamesE.at(nERecord))
        {
            QDomNode CType =
XML_Check::FindChildXml(g_aaCNamesString.at(nERecord).at(nRPtype
),g_CCurrentParentNode);
            if(CType.childNodes().at(0).nodeValue() ==
g_aCTypeVal.at(VType_B))
            {
                CChild =
XML_Check::CreateBit(g_CXDoc,&g_aaCNamesString,&g_aCElementNames
E,&g_aCTypeVal);

                QDomNode CBits =
XML_Check::FindChildXml(g_aaCNamesString.at(nERecord).at(nRPbits
),g_CCurrentParentNode);
                CBits.appendChild(CChild);

            }
            else
            {
                bNeedToUpdateTree = false;
                QErrorMessage CErr;

CErr.showMessage(g_aCErrorMessages.at(EMUserAddToNonBitRecord),g
_aCErrorMessages.at(EMError));
                CErr.exec();
            }
        }
        else if (g_CCurrentParentNode.nodeName() ==
g_aCElementNamesE.at(nEBit))
        {
            bNeedToUpdateTree = false;
            QErrorMessage CErr;

CErr.showMessage(g_aCErrorMessages.at(EMUserAddToBit),g_aCErrorM
essages.at(EMError));
            CErr.exec();
        }

        if(bNeedToUpdateTree)
        {
            this-
>UpdateTree(g_CXDoc,&g_aCElementNamesE,&g_aCElementNamesR,&g_aaC
NamesString);
            if(bNeedToUpdateTable)
            {
```

```
                tableTree->blockSignals(true);
                tableTree-
>UpdateTable(g_aCXmlParams,g_CCurrentParentNode.nodeName(),&g_aC
ElementNamesE);
                tableTree->blockSignals(false);
            }
        }
    }
}

void XML_Editor::on_pushButton_clicked()
{
    bool bIsXDocNull = false;
    bool bNeedToCreate = true;

    if(g_CXDoc->isNull())
    {
        bIsXDocNull = true;
        g_CXDoc = new QDomDocument();
    }
    else
    {
        if(g_bAllowDialogWindow)
        {
            QMessageBox CMsgBox;

            CMsgBox.setWindowTitle(QString("Создание
модели").toUtf8());
            CMsgBox.setText(QString("При создании новой модели
старая модель будет удалена.").toUtf8());
            CMsgBox.setInformativeText(QString("Вы уверены, что
хотите продолжить?").toUtf8());
            CMsgBox.setIcon(QMessageBox::Question);
            CMsgBox.setStandardButtons(QMessageBox::Yes |
QMessageBox::No);

            int nRes = CMsgBox.exec();
            if(nRes == QMessageBox::No)
                bNeedToCreate = false;
        }
    }

    if(bNeedToCreate)
    {
        if(!bIsXDocNull)
            g_CXDoc->removeChild(g_CXDoc->documentElement());

        QDomNode CUnits =
XML_Check::CreateModel(g_CXDoc,&g_aaCNamesString,&g_aCElementNam
esE);

        g_CXDoc->appendChild(CUnits);
```

```
        g_CCurrentParentNode = CUnits;
        g_aCXmlParams =
CreateParamList(g_aCXmlParams,g_CCurrentParentNode,g_CCurrentPar
entNode.nodeName(),&g_aaCNamesString,&g_aCElementNamesE);
        this-
>UpdateTree(g_CXDoc,&g_aCElementNamesE,&g_aCElementNamesR,&g_aaC
NamesString);
        g_CSelectedTreeItem = ui->treeWidget->topLevelItem(0);
        tableTree->blockSignals(true);
        tableTree-
>UpdateTable(g_aCXmlParams,g_CCurrentParentNode.nodeName(),&g_aC
ElementNamesE);
        tableTree->blockSignals(false);
    }
}

void XML_Editor::on_pushButton_5_clicked()
{
    bool bIsCurrentNodeNull = g_CCurrentParentNode.isNull();
    if(!bIsCurrentNodeNull &&
g_CCurrentParentNode.nodeName()!=g_aCElementNamesE.at(nEModel))
    {
        int nRes = 0;
        if(g_bAllowDialogWindow)
        {
            QMessageBox CMsgBox;

            CMsgBox.setWindowTitle(QString("Удаление
элемента").toUtf8());
            CMsgBox.setText(QString("Вы действительно хотите
удалить элемент?").toUtf8());
            CMsgBox.setIcon(QMessageBox::Question);
            CMsgBox.setStandardButtons(QMessageBox::Yes |
QMessageBox::No);
            nRes = CMsgBox.exec();
        }

        if(!g_bAllowDialogWindow || nRes == QMessageBox::Yes)
        {
            bool bNeedToUpdateMaxSizeRecord = false;
            bool bNeedToUpdateCountUnit = false;
            if(g_CCurrentParentNode.nodeName() ==
g_aCElementNamesE.at(nEUnit) ||
               g_CCurrentParentNode.nodeName() ==
g_aCElementNamesE.at(nERecord))
            {
                bNeedToUpdateMaxSizeRecord = true;
                if(g_CCurrentParentNode.nodeName() ==
g_aCElementNamesE.at(nEUnit))
                    bNeedToUpdateCountUnit = true;
            }
```

```
g_CCurrentParentNode.parentNode().removeChild(g_CCurrentParentNo
de);

UpdateTree(g_CXDoc,&g_aCElementNamesE,&g_aCElementNamesR,&g_aaCN
amesString);
            tableTree->ClearTable();
            if(bNeedToUpdateCountUnit)

XML_Check::CheckUnitsCount(g_CXDoc,&g_aaCNamesString,&g_aCElemen
tNamesE);
            if(bNeedToUpdateMaxSizeRecord)

XML_Check::CalcMaxSizeRecord(g_CXDoc,&g_aaCNamesString,&g_aCElem
entNamesE);
        }
    }
}

void XML_Editor::on_treeWidget_itemClicked(QTreeWidgetItem
*item, int column)
{

}


void XML_Editor::ShowContextMenu(const QPoint& pos)
{
    for(int i = 0; i < m_aCActions.count();i++)
        m_aCActions.at(i)->setVisible(false);

    if(!g_CCurrentParentNode.isNull())
    {
        QPoint CGlobalPos = ui->treeWidget->mapToGlobal(pos);

        bool bIsModel = g_CCurrentParentNode.nodeName() ==
g_aCElementNamesE.at(nEModel);
        bool bIsRecord = g_CCurrentParentNode.nodeName() ==
g_aCElementNamesE.at(nERecord);
        bool bIsBit = g_CCurrentParentNode.nodeName() ==
g_aCElementNamesE.at(nEBit);


//        QMenu CMenu;
//        QAction *CAddChild = new QAction("Добавить потомка");
//
connect(CAddChild,SIGNAL(triggered()),this,SLOT(on_pushButton_4_
clicked()));

        if(bIsModel)
            m_aCActions.at(A_AddChild)->setVisible(true);
        else
        {
```

```
            QTreeWidgetItem *CParent = g_CSelectedTreeItem-
>parent();
            if(CParent->indexOfChild(g_CSelectedTreeItem) > 0)
                m_aCActions.at(A_ReplaceHigher)-
>setVisible(true);
            if(CParent->indexOfChild(g_CSelectedTreeItem) <
CParent->childCount() - 1)
                m_aCActions.at(A_ReplaceLower)-
>setVisible(true);

            // Модуль или запись
            if ( !bIsBit )
            {
                // Задаём соответствующий набор параметров
контекстного меню
                m_aCActions.at(A_AddHigher)->setVisible(true);
                m_aCActions.at(A_AddLower)->setVisible(true);
                m_aCActions.at(A_AddChild)->setVisible(true);
                m_aCActions.at(A_Remove)->setVisible(true);

                // Если запись
                if ( bIsRecord )
                {
                    // Если тип не битовый, то удаляем свойство
"добавить потомка"
                    QDomNode CType =
XML_Check::FindChildXml(g_aaCNamesString.at(nERecord).at(nRPtype
),g_CCurrentParentNode);

                    bool bIsType_b =
CType.childNodes().at(0).nodeValue() == g_aCTypeVal.at(VType_B);

                    if ( !bIsType_b )
                        m_aCActions.at(A_AddChild)-
>setVisible(false);
                }

            }
            // Если бит, то устанавливаем в контекстном меню
соответствующий набор параметров
            else
            {
                m_aCActions.at(A_AddHigher)->setVisible(true);
                m_aCActions.at(A_AddLower)->setVisible(true);
                m_aCActions.at(A_Remove)->setVisible(true);
            }

        }

        m_CMenu.exec(CGlobalPos);
    }
}
```

```
void XML_Editor::ReplaceElementHigher()
{
    // Родительский элемент выбранного эл-та XML
    QDomNode CParentXmlNode = g_CCurrentParentNode.parentNode();

    // Предыдущий эл-т XML
    QDomNode CPrevNode = g_CCurrentParentNode;

    while(true)
    {
        CPrevNode = CPrevNode.previousSibling();
        if(CPrevNode.nodeName() ==
g_CCurrentParentNode.nodeName())
            break;
    }

    // Меняем эл-ты местами
    CParentXmlNode.insertBefore(g_CCurrentParentNode,CPrevNode);


UpdateTree(g_CXDoc,&g_aCElementNamesE,&g_aCElementNamesR,&g_aaCN
amesString);

}

void XML_Editor::ReplaceElementLower()
{
    // Родительский элемент выбранного эл-та XML
    QDomNode CParentXmlNode = g_CCurrentParentNode.parentNode();

    // Следующий эл-т XML
    QDomNode CNextNode = g_CCurrentParentNode;

    while(true)
    {
        CNextNode = CNextNode.nextSibling();
        if(CNextNode.nodeName() ==
g_CCurrentParentNode.nodeName())
            break;
    }

    // Меняем эл-ты местами
    CParentXmlNode.insertAfter(g_CCurrentParentNode,CNextNode);


UpdateTree(g_CXDoc,&g_aCElementNamesE,&g_aCElementNamesR,&g_aaCN
amesString);
}

void XML_Editor::AddElementHigher()
{
    // Родительский элемент выбранного эл-та XML
    QDomNode CParentXmlNode = g_CCurrentParentNode.parentNode();
```

```
    // Новый элемент
    QDomNode CNewXml;

    bool bIsModel  = g_CCurrentParentNode.nodeName() ==
g_aCElementNamesE.at(nEModel);
    bool bIsUnit   = g_CCurrentParentNode.nodeName() ==
g_aCElementNamesE.at(nEUnit);
    bool bIsRecord = g_CCurrentParentNode.nodeName() ==
g_aCElementNamesE.at(nERecord);

    // Если выбранный элемент это модуль
    if ( bIsUnit )
        CNewXml =
XML_Check::CreateUnit(g_CXDoc,&g_aaCNamesString,&g_aCElementName
sE);
    else
    {
        // Если выбранный элемент это запись, то вызываем
функцию добавления записи, иначе добавляем бит
        if(bIsRecord)
            CNewXml =
XML_Check::CreateRecord(g_CXDoc,&g_aaCNamesString,&g_aCElementNa
mesE,&g_aCTypeVal);
        else
            CNewXml =
XML_Check::CreateBit(g_CXDoc,&g_aaCNamesString,&g_aCElementNames
E,&g_aCTypeVal);
    }

    // Размещаем новый элемент перед выбранным
    CParentXmlNode.insertBefore(CNewXml,g_CCurrentParentNode);
    // Обновляем дерево

UpdateTree(g_CXDoc,&g_aCElementNamesE,&g_aCElementNamesR,&g_aaCN
amesString);

    // Если текущий элемент - модель, пересчитать количество
модулей
    if(bIsUnit)

XML_Check::CheckUnitsCount(g_CXDoc,&g_aaCNamesString,&g_aCElemen
tNamesE);
}

void XML_Editor::AddElementLower()
{
    // Родительский элемент выбранного эл-та XML
    QDomNode CParentXmlNode = g_CCurrentParentNode.parentNode();

    // Новый элемент
    QDomNode CNewXml;
```

```
    bool bIsModel  = g_CCurrentParentNode.nodeName() ==
g_aCElementNamesE.at(nEModel);
    bool bIsUnit   = g_CCurrentParentNode.nodeName() ==
g_aCElementNamesE.at(nEUnit);
    bool bIsRecord = g_CCurrentParentNode.nodeName() ==
g_aCElementNamesE.at(nERecord);

    // Если выбранный элемент это модуль
    if ( bIsUnit )
        CNewXml =
XML_Check::CreateUnit(g_CXDoc,&g_aaCNamesString,&g_aCElementName
sE);
    else
    {
        // Если выбранный элемент это запись, то вызываем
функцию добавления записи, иначе добавляем бит
        if(bIsRecord)
            CNewXml =
XML_Check::CreateRecord(g_CXDoc,&g_aaCNamesString,&g_aCElementNa
mesE,&g_aCTypeVal);
        else
            CNewXml =
XML_Check::CreateBit(g_CXDoc,&g_aaCNamesString,&g_aCElementNames
E,&g_aCTypeVal);
    }

    // Размещаем новый элемент перед выбранным
    CParentXmlNode.insertAfter(CNewXml,g_CCurrentParentNode);
    // Обновляем дерево

UpdateTree(g_CXDoc,&g_aCElementNamesE,&g_aCElementNamesR,&g_aaCN
amesString);

    // Если текущий элемент - модель, пересчитать количество
модулей
    if(bIsUnit)

XML_Check::CheckUnitsCount(g_CXDoc,&g_aaCNamesString,&g_aCElemen
tNamesE);
}

void XML_Editor::ClearDocument(QDomDocument* CXDoc, QStringList
*aCElementNamesE,QList<QStringList> *aaCNamesString)
{
    // Цикл по кол-ву элементов в корне документа
    for(int i = 0; i < CXDoc->childNodes().count(); i++)
    {
        // Если не декларация и не Units, удалить
        bool bIsDeclaration = CXDoc-
>childNodes().at(i).isDocumentType();
        bool bIsModel = CXDoc->childNodes().at(i).nodeName() ==
aCElementNamesE->at(nEModel);
```

```
        if ( !bIsDeclaration && !bIsModel )
        {
            CXDoc->removeChild(CXDoc->childNodes().at(i));
            i--;
        }
        // Если Units, вызвать функцию очистки Units
        else if ( bIsModel )
            ClearUnits(CXDoc-
>childNodes().at(i),aCElementNamesE,aaCNamesString);
    }
}

void XML_Editor::ClearUnits(QDomNode CUnits, QStringList
*aCElementNamesE,QList<QStringList> *aaCNamesString)
{
    // Массив свойств модели
    QDomNode aCXArray[aaCNamesString->at(nEModel).count()];

    // Цикл по кол-ву дочерних элементов корневого эл-та
    for(int i = 0; i < CUnits.childNodes().count(); i++)
    {
        // Если мусор, то удалить
        bool bIsModule = CUnits.childNodes().at(i).nodeName() ==
aCElementNamesE->at(nEUnit);
        bool bIsRubbish = !aaCNamesString-
>at(nEModel).contains(CUnits.childNodes().at(i).nodeName());

        if ( !bIsModule && bIsRubbish )
        {
            CUnits.removeChild(CUnits.childNodes().at(i));
            i--;
        }
        else
        {
            // Если модуль, то вызвать функцию проверки модуля
            if ( bIsModule )

ClearUnit(CUnits.childNodes().at(i),aCElementNamesE,aaCNamesStri
ng);

            else
            {
                // Цикл по кол-ву имён свойств модели
                for(int j = 0; j < aaCNamesString-
>at(nEModel).length();j++)
                {
                    // Если имя эл-та соответствует текущему
имени свойства, занести элемент в массив свойств
                    bool bIsNameOk =
CUnits.childNodes().at(i).nodeName() == aaCNamesString-
>at(nEModel).at(j);

                    if(!bIsNameOk) continue;
```

```
                    aCXArray [ j ] = CUnits.childNodes().at(i);
                }
            }
        }
    }

    // Цикл упорядочивания структуры
    for ( int i = 0; i < aaCNamesString->at(nEModel).count();
i++ )

CUnits.insertBefore(aCXArray[i],CUnits.childNodes().at(i));
}

void XML_Editor::ClearUnit(QDomNode CUnit, QStringList
*aCElementNamesE, QList<QStringList> *aaCNamesString)
{
    // Массив свойств модуля
    QDomNode aCXArray[aaCNamesString->at(nEUnit).count()];

    // Цикл по кол-ву дочерних элементов модуля
    for(int i = 0; i < CUnit.childNodes().count(); i++)
    {
        // Если мусор, то удалить
        bool bIsRubbish = !aaCNamesString-
>at(nEUnit).contains(CUnit.childNodes().at(i).nodeName());
        if ( bIsRubbish )
        {
            CUnit.removeChild(CUnit.childNodes().at(i));
            i--;
        }

        else
        {
            // Цикл по кол-ву имён свойств модуля
            for(int j = 0; j < aaCNamesString-
>at(nEUnit).count(); j++)
            {
                // Если имя эл-та не соответствует текущему
имени свойства
                bool bIsParam =
CUnit.childNodes().at(i).nodeName() == aaCNamesString-
>at(nEUnit).at(j);
                if ( !bIsParam ) continue;

                // Занести элемент в массив свойств
                aCXArray [ j ] = CUnit.childNodes().at(i);

                // Если это поле records, вызвать функцию
очистки records
                bool bIsRecords = j == nUPrecords;
                if ( !bIsRecords ) continue;
```

```
ClearRecords(CUnit.childNodes().at(i),aCElementNamesE,aaCNamesSt
ring);

            }
        }
    }
    for ( int i = 0; i < aaCNamesString->at(nEUnit).count(); i++
)

CUnit.insertBefore(aCXArray[i],CUnit.childNodes().at(i));
}

void XML_Editor::ClearRecords(QDomNode CRecords, QStringList
*aCElementNamesE, QList<QStringList> *aaCNamesString)
{
    // Цикл по кол-ву дочерних элементов records
    for ( int i = 0; i < CRecords.childNodes().count(); i++ )
    {
        // Если запись, то вызвать функцию проверки записи,
иначе удалить
        bool bIsRecord = CRecords.childNodes().at(i).nodeName()
== aCElementNamesE->at(nERecord);
        if ( bIsRecord )

ClearRecord(CRecords.childNodes().at(i),aCElementNamesE,aaCNames
String);

        else
        {
            CRecords.removeChild(CRecords.childNodes().at(i));
            i--;
        }
    }
}

void XML_Editor::ClearRecord(QDomNode CRecord, QStringList
*aCElementNamesE, QList<QStringList> *aaCNamesString)
{
    // Массив свойств записи
    QDomNode aCXArray[aaCNamesString->at(nERecord).count() - 2];

    // Цикл по кол-ву дочерних элементов записи
    for ( int i = 0; i < CRecord.childNodes().count(); i++ )
    {
        // Если мусор, то удалить
        bool bIsRubbish = !aaCNamesString-
>at(nERecord).contains(CRecord.childNodes().at(i).nodeName());
        if ( bIsRubbish )
        {
            CRecord.removeChild(CRecord.childNodes().at(i));
            i--;
        }
```

```
        else
        {
            // Если bits, то вызвать функцию проверки bits
            bool bIsBits = CRecord.childNodes().at(i).nodeName()
== aaCNamesString->at(nERecord).at(nRPbits);
            if ( bIsBits )

ClearBits(CRecord.childNodes().at(i),aCElementNamesE,aaCNamesStr
ing);
            else
            {
                // Цикл по кол-ву имён свойств записи
                for ( int j = 0; j < aaCNamesString-
>at(nERecord).count() - 2; j++ )
                {
                    bool bIsParam =
CRecord.childNodes().at(i).nodeName() == aaCNamesString-
>at(nERecord).at(j);
                    // Если имя эл-та соответствует текущему
имени свойства, занести элемент в массив свойств
                    if ( !bIsParam ) continue;
                    aCXArray [ j ] = CRecord.childNodes().at(i);
                }
            }
        }
    }

    // Цикл упорядочивания структуры
    for ( int i = 0; i < aaCNamesString->at(nERecord).count() -
2; i++ )

CRecord.insertBefore(aCXArray[i],CRecord.childNodes().at(i));
}

void XML_Editor::ClearBits(QDomNode CBits, QStringList
*aCElementNamesE, QList<QStringList> *aaCNamesString)
{
    // Цикл по кол-ву дочерних элементов bits
    for ( int i = 0; i < CBits.childNodes().count(); i++ )
    {
        // Если бит, то вызвать функцию проверки бита, иначе
удалить
        bool bIsBit = CBits.childNodes().at(i).nodeName() ==
aCElementNamesE->at(nEBit);
        if ( bIsBit )

ClearBit(CBits.childNodes().at(i),aCElementNamesE,aaCNamesString
);
        else
        {
            CBits.removeChild(CBits.childNodes().at(i));
            i--;
```

```
        }
    }
}

void XML_Editor::ClearBit(QDomNode CBit, QStringList
*aCElementNamesE, QList<QStringList> *aaCNamesString)
{
    // Массив свойств бита
    QDomNode aCXArray[aaCNamesString->at(nEBit).count() - 1];

    // Цикл по кол-ву дочерних элементов бита
    for ( int i = 0; i < CBit.childNodes().count(); i++ )
    {
        // Если мусор, то удалить
        bool bIsRubbish = !aaCNamesString-
>at(nEBit).contains(CBit.childNodes().at(i).nodeName());
        if ( bIsRubbish )
        {
            CBit.removeChild(CBit.childNodes().at(i));
            i--;
        }

        else
        {
            // Цикл по кол-ву имён свойств бита
            for ( int j = 0; j < aaCNamesString-
>at(nEBit).count() - 1; j++ )
            {
                // Если имя эл-та соответствует текущему имени
свойства, занести элемент в массив свойств
                bool bIsParam =
CBit.childNodes().at(i).nodeName() == aaCNamesString-
>at(nEBit).at(j);

                if ( !bIsParam ) continue;
                aCXArray [ j ] = CBit.childNodes().at(i);
            }
        }
    }
    // Цикл упорядочивания структуры
    for ( int i = 0; i < aaCNamesString->at(nEBit).count() - 1;
i++ )
        CBit.insertBefore(aCXArray[i],CBit.childNodes().at(i));
}
```

Приложение В

Пример XML-документа

```
<Units>
 <CountByteHeader>0</CountByteHeader>
 <CountUnit>2</CountUnit>
 <MaxSizeRecord>1</MaxSizeRecord>
 <Unit>
  <codeOperation>0</codeOperation>
  <information>1</information>
  <records>
   <record>
    <offset>0</offset>
    <size>1</size>
    <string>1</string>
    <type>b</type>
    <bits>
     <bit>
      <offset>0</offset>
      <size>1</size>
      <string>1</string>
      <type>d</type>
      <goodvalue>1</goodvalue>
     </bit>
    </bits>
   </record>
   <record>
    <offset>0</offset>
    <size>0</size>
    <string>2</string>
    <type>d</type>
   </record>
  </records>
 </Unit>
 <Unit>
  <codeOperation>0</codeOperation>
  <information>2</information>
  <records/>
 </Unit>
</Units>
```

Приложение С

UML-диаграммы приложения

Приложение С.1 – UML-диаграмма классов

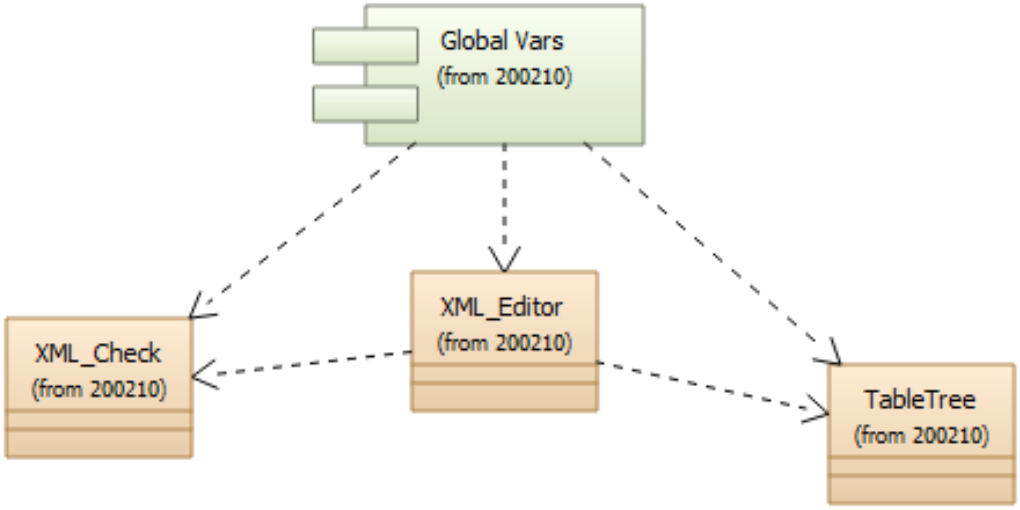


Рисунок 55 – UML- диаграмма классов

Приложение С.2 – UML-диаграмма вариантов использования

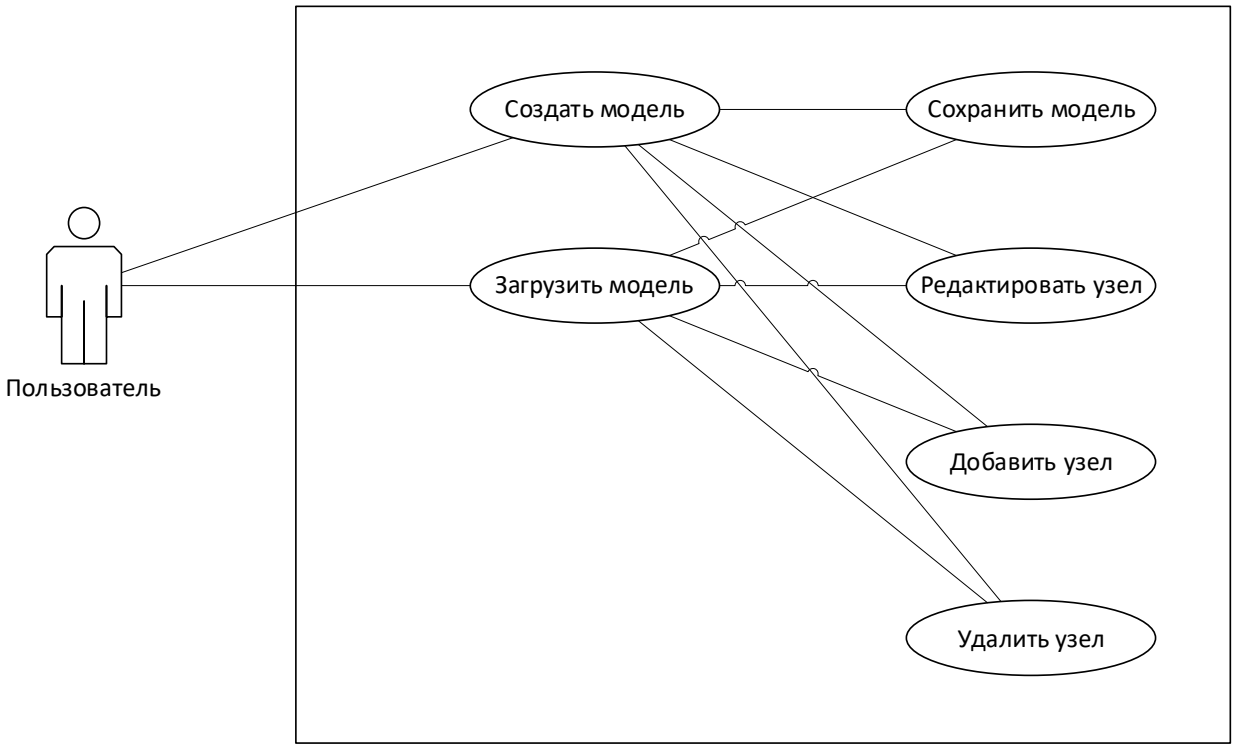


Рисунок 56 – UML- диаграмма вариантов использования

Приложение С.3 – UML-диаграмма последовательности

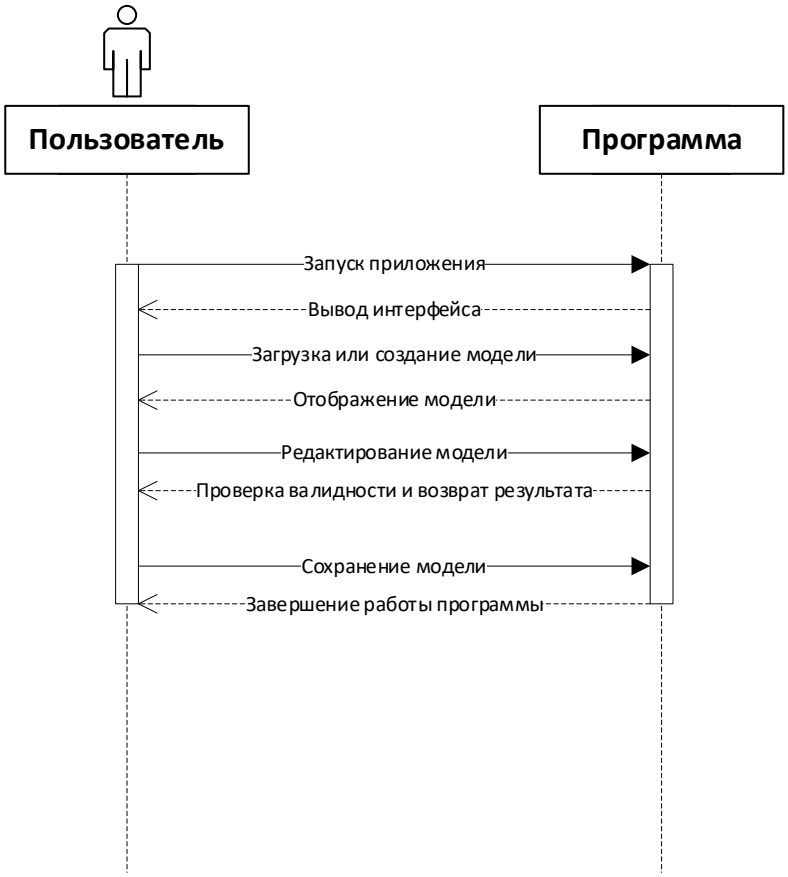


Рисунок 57 – UML- диаграмма последовательности

Приложение D

Результаты работы программы

Приложение D.1 – Результат загрузки файла

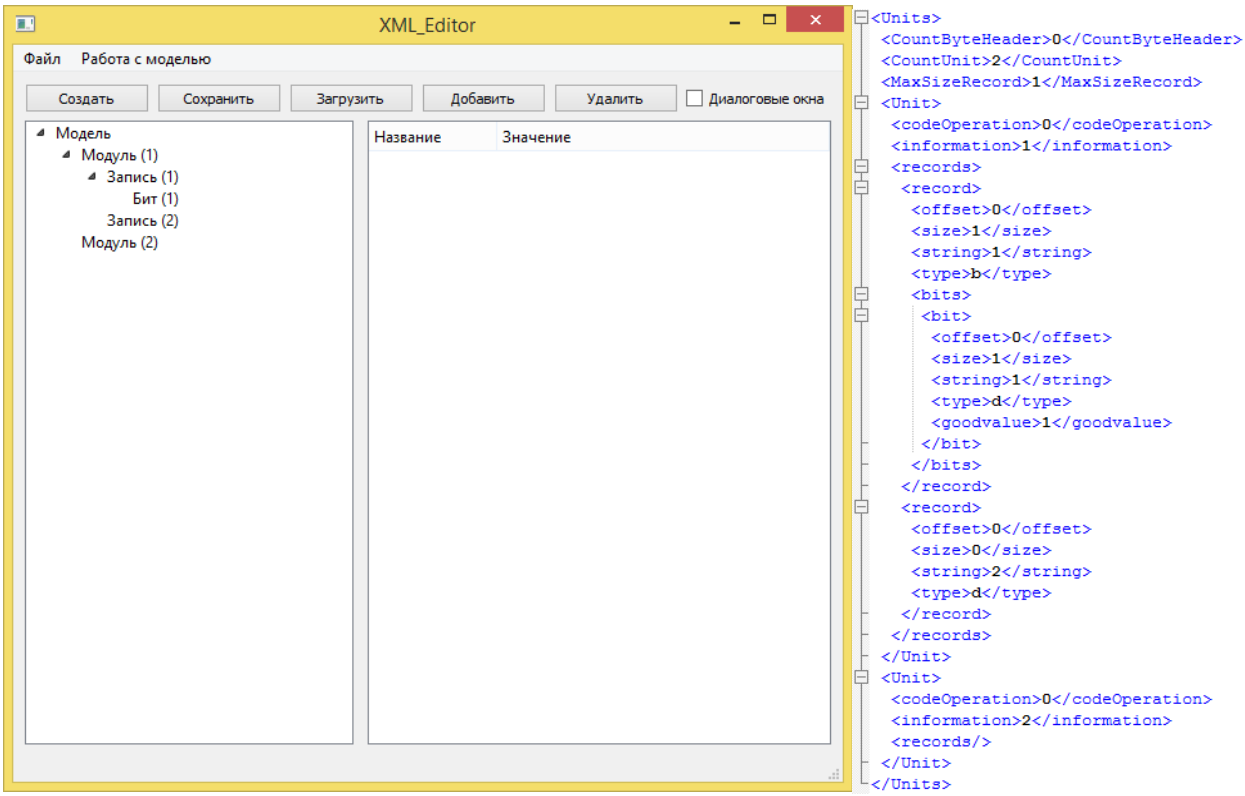


Рисунок 58 - Результат загрузки файла

Приложение D.2 – Добавление элемента

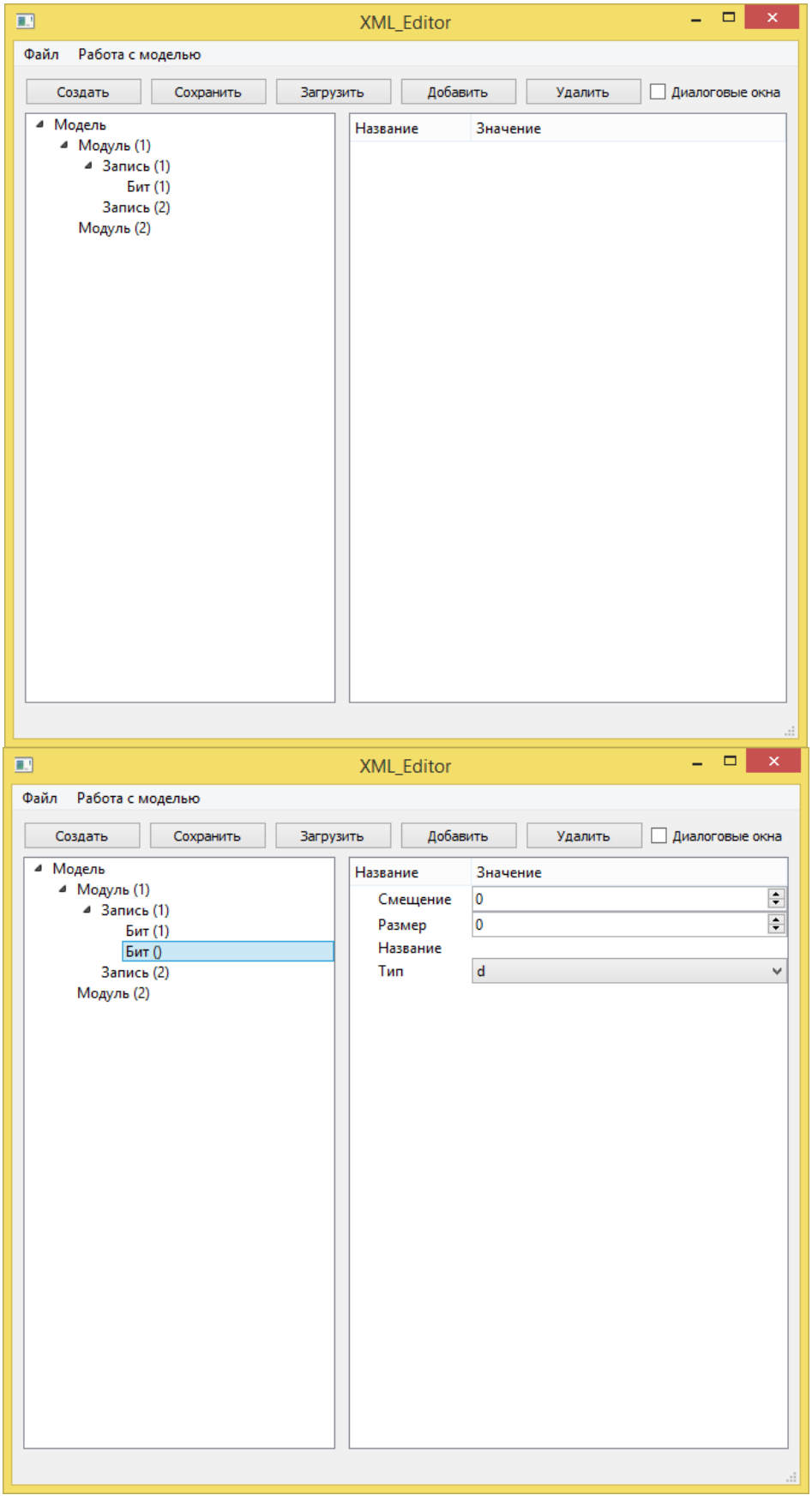


Рисунок 59 – Добавление элемента

Приложение D.3 – Редактирование элемента

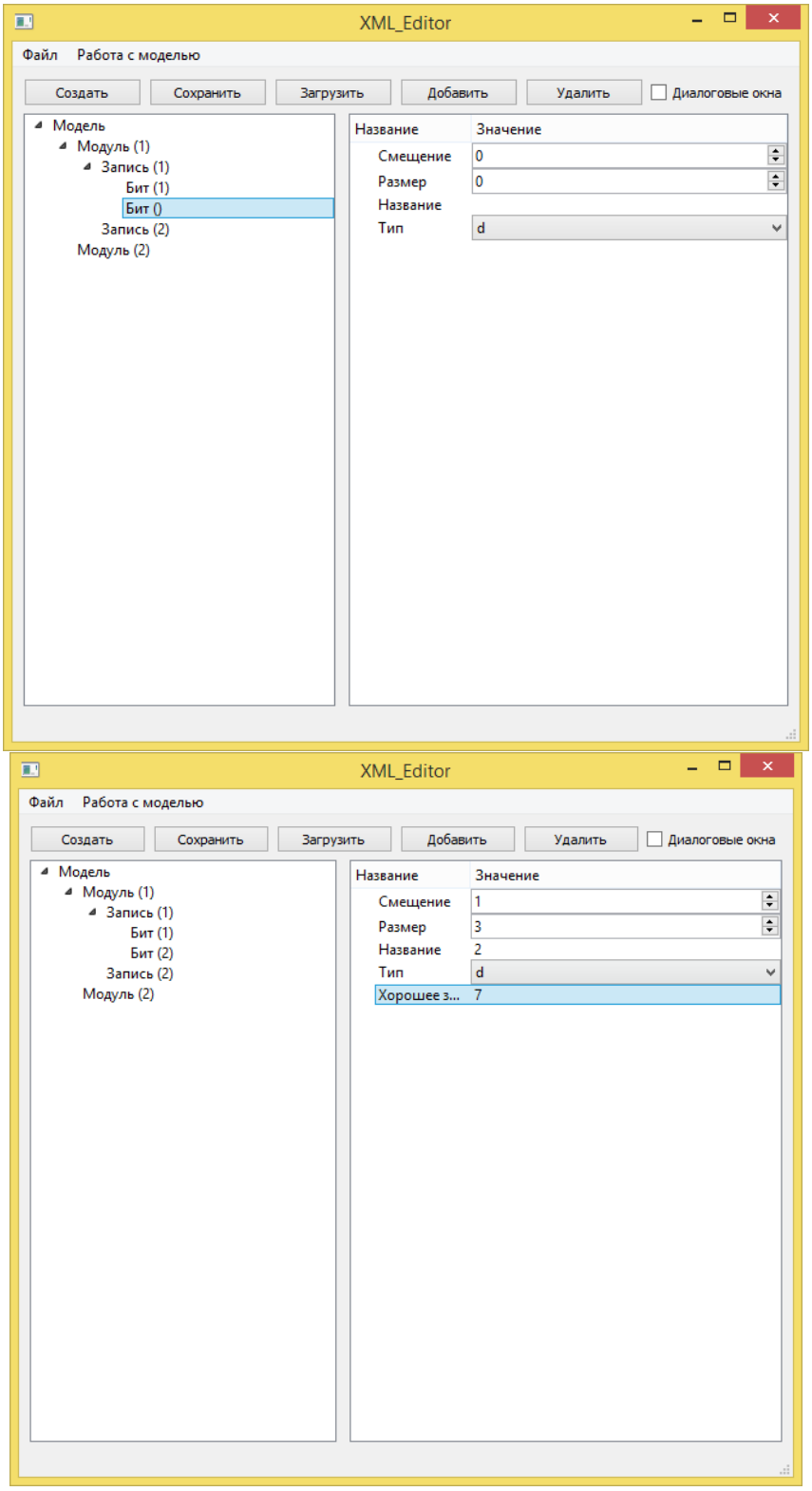


Рисунок 60 – Редактирование элемента

Приложение D.4 – Сохранение документа

```
<Units>
 <CountByteHeader>0</CountByteHeader>
 <CountUnit>2</CountUnit>
 <MaxSizeRecord>1</MaxSizeRecord>
 <Unit>
  <codeOperation>0</codeOperation>
  <information>1</information>
  <records>
   <record>
    <offset>0</offset>
    <size>1</size>
    <string>1</string>
    <type>b</type>
    <bits>
     <bit>
      <offset>0</offset>
      <size>1</size>
      <string>1</string>
      <type>d</type>
      <goodvalue>1</goodvalue>
     </bit>
     <bit>
      <offset>1</offset>
      <size>3</size>
      <string>2</string>
      <type>d</type>
      <goodvalue>7</goodvalue>
     </bit>
    </bits>
   </record>
   <record>
    <offset>0</offset>
    <size>0</size>
    <string>2</string>
    <type>d</type>
   </record>
  </records>
 </Unit>
 <Unit>
  <codeOperation>0</codeOperation>
  <information>2</information>
  <records/>
 </Unit>
</Units>
```

Рисунок 61 – Сохранение документа

Приложение Е

Презентация

Синтаксический редактор телеметрической информации

Автор работы: студент группы 16ВВ2 Прокуров Д.А.

Научный руководитель: д.т.н., доцент Федюнин Р.Н.

Цель и объект работы

Объектом разработки является приложение, обеспечивающее выполнение следующих функций:

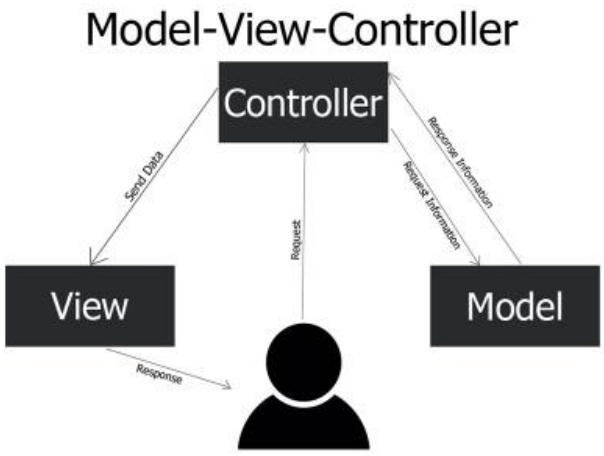
- Проверка;
- Создание;
- Редактирование
- Загрузка
- Сохранение XML-документа, представляющего собой описание телеметрической информации. Также необходимо реализовать API для взаимодействия с пользователем.

Язык XML

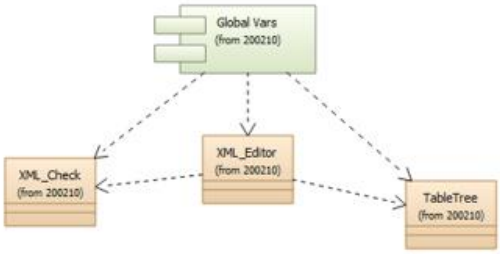
Язык XML – один из самых популярных и распространённых языков разметки. Часто используется для представления иерархических структур данных

```
<?xml version="1.0" encoding="iso-8859-8" standalone="yes" ?>
<CURRENCIES>
  <LAST_UPDATE>2004-07-29</LAST_UPDATE>
  <CURRENCY>
    <NAME>dollar</NAME>
    <UNIT>1</UNIT>
    <CURRENCYCODE>USD</CURRENCYCODE>
    <COUNTRY>USA</COUNTRY>
    <RATE>4.527</RATE>
    <CHANGE>0.044</CHANGE>
  </CURRENCY>
  <CURRENCY>
    <NAME>euro</NAME>
    <UNIT>1</UNIT>
    <CURRENCYCODE>EUR</CURRENCYCODE>
    <COUNTRY>European Monetary Union</COUNTRY>
    <RATE>5.4417</RATE>
    <CHANGE>-0.013</CHANGE>
  </CURRENCY>
</CURRENCIES>
```

Архитектура MVC



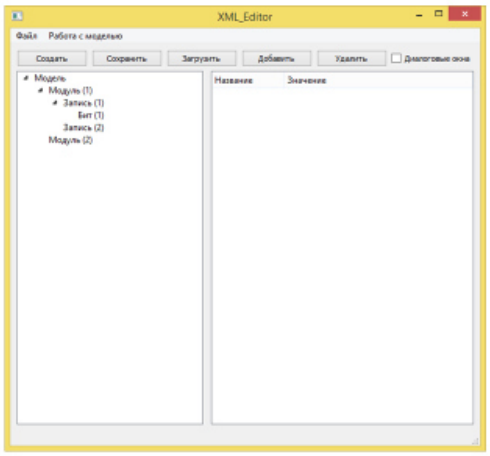
Разработка приложения



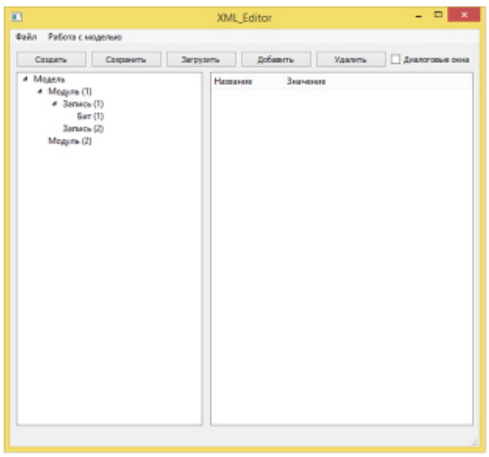
UML-диаграмма классов

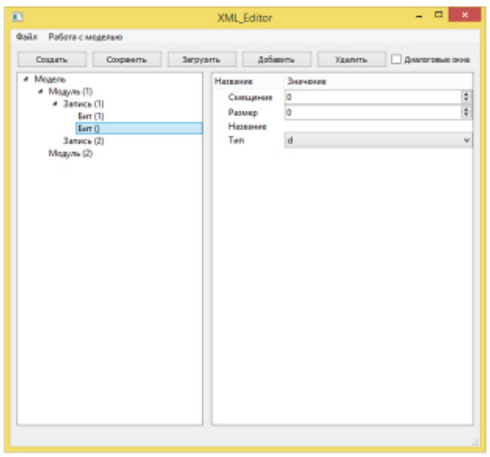
Загрузка документа

```
<Units>
 <CountByteHeader>0</CountByteHeader>
 <CountUnit>2</CountUnit>
 <MaxSizeRecord>1</MaxSizeRecord>
 <Unit>
  <codeOperation>0</codeOperation>
  <information>1</information>
  <records>
   <record>
    <offset>0</offset>
    <size>1</size>
    <string>1</string>
    <type>b</type>
    <bits>
     <bit>
      <offset>0</offset>
      <size>1</size>
      <string>1</string>
      <type>d</type>
      <goodvalue>1</goodvalue>
     </bit>
    </bits>
   </record>
   <record>
    <offset>0</offset>
    <size>0</size>
    <string>2</string>
    <type>d</type>
   </record>
  </records>
 </Unit>
 <Unit>
  <codeOperation>0</codeOperation>
  <information>2</information>
  <records/>
 </Unit>
</Units>
```

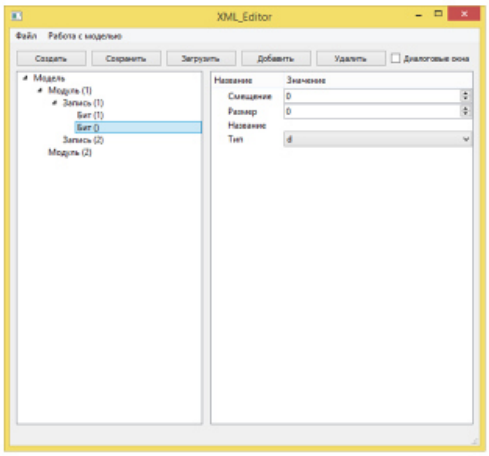


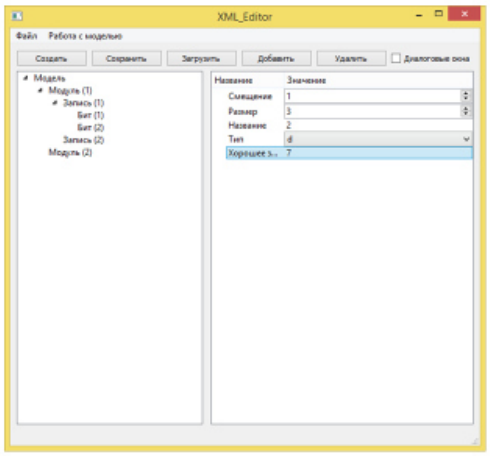
Добавление элемента





Редактирование элемента





Сохранение документа

Спасибо за внимание